- 全面详解Apache Cordova移动跨平台开发框架与开发方法
- 布局移动网站和移动应用，切中目前企业最热的开发需求
- 低成本，快速开发，还能跨平台，利用最好最流行的技术进行实战演练





Target Multiple Platforms with One Code Base

Apache Cordova 移动应用开发实战

王亚飞 王洪飞 编著



清华大学出版社

跨平台移动开发丛书



Apache Cordova 移动应用开发实战

王亚飞　王洪飞　编著

清華大学出版社
北　京

内 容 简 介

Cordova 是一款优秀的移动跨平台开发框架，开发者通过它能够快速地将 Web 应用打包成在各个平台上运行的本地 APP。

本书分 3 篇共 16 章，第一篇是入门篇，包括了 Cordova 的小伙伴们、在安卓和 iOS 开发环境下的配置、对 HTML 5 前景的简单介绍。第二篇是基础知识篇，包含了本地事件设备信息、通讯录、加速度传感器、设备传感器、音频、文件、多媒体资源等 Cordova 中 API 的实例。第三篇是项目实战篇，包括简单的游戏（Flappy Bird）、新闻客户端，以及结合 jQuery Mobile 制作的号码本。

本书内容详尽、实例丰富，适合 Cordova 跨平台 APP 开发的初学者，尤其是在校学生，以及有意在互联网时代捞到第一桶金的创业者。

图书在版编目（CIP）数据

Apache Cordova 移动应用开发实战/王亚飞，王洪飞编著. —北京：清华大学出版社，2017

（跨平台移动开发丛书）

ISBN 978-7-302-47067-0

I. ①A… II. ①王… ②王… III. ①移动终端－应用程序－程序设计 IV. ①TN929.53

中国版本图书馆 CIP 数据核字（2017）第 114100 号

责任编辑：夏毓彦
封面设计：王　翔
责任校对：闫秀华
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　**邮　　编：**100084
社 总 机：010-62770175　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：190mm×260mm　**印　张：**18.75　**字　数：**480 千字
版　　次：2017 年 7 月第 1 版　**印　次：**2017 年 7 月第 1 次印刷
印　　数：1~3500
定　　价：69.00 元

产品编号：073281-01

前　言

Cordova 是一款简单、易上手的移动跨平台开发框架，也是 Adobe 公司极力推荐的一款开发框架。它具有开发效率高、上手简单以及一次部署七大平台全部兼容等优点。遗憾的是由于国内仍然缺少一套完整的 Cordova 教程，使得它虽然已经被许多开发者认识，却始终难以真正推广开来。本书的出现将弥补这一遗憾。本书全面地介绍了 Cordova 的 API 使用、Cordova 插件的编写方法、利用 JavaScript 获取信息的方法，以及 Cordova 与 jQuery Mobile 相互配合使用的方法，力求让本书的读者能举一反三，并最终实现自己的梦想。

本书特色

1. 内容丰富，知识全面

本书采用从易到难、实例结合理论的方式进行讲解，内容几乎涉及了 Cordova 的各个方面。

2. 循序渐进，由浅入深

为了方便读者学习，本书首先介绍了一些基本常识，如什么是 HTML 5 以及 Cordova 配置等内容，然后开始使用 Cordova 中的 API 实现一些小的例子，最终过渡到真正利用 Cordova 实现完整的应用。

3. 格式统一，讲解规范

书中每个知识点都给出了详尽的操作示例供读者参考，通过实践可以使读者更清晰地了解每个知识点的细节，提高学习效率。

4. 内容详尽，方便学习

虽然 Cordova 能够实现跨平台的功能，但是目前它确实还有不够完善的地方，许多读者在学习时可能会遇到不知名的“错误”而导致中途放弃。本书根据作者的多年经验指出一些可能由于 Cordova 或者安卓系统本身的原因造成的错误，力求使读者少走弯路、高效学习。

5. 案例精讲，深入剖析

本书的每个知识点都是通过实例来介绍，使读者在学习每个知识点时都能够通过动手来加深印象。本书第三篇的三个项目使读者能够有机会理解到真实项目和知识点的区别，并切实掌握利用 Cordova 进行应用开发的精髓。

本书结构

本书分 3 篇共 16 章，主要章节规划如下。

第一篇（第 1 章~第 3 章）入门篇

在学习之前进行一些前置知识的介绍，包括：什么是 Cordova、怎样使用 Cordova 以及跨平台的 HTML 5 等内容。本篇最后总览了 Cordova 所提供的 API，让读者在学习具体知识点之前先对 Cordova 有一个大概的了解。

第二篇（第 4 章~第 13 章）基础知识篇

介绍了 Cordova 中的 API，包括事件管理、本地存储、音视频处理、文件管理等内容，并结合作者本人的经验给出了使用建议。

第三篇（第 14 章~第 16 章）项目实战

本篇学习三个利用 Cordova 实现的项目：Flappy Bird（像素鸟）游戏、新闻客户端、号码本，能够让读者从学习知识转化到项目实战中去，真正将所学的知识加以应用。

本书读者

- Android、iOS 移动产品开发人员
- HTML 5、HTML 移动产品开发人员
- 跨平台移动开发初学者
- 有好的想法但是由于技术限制难以实现的移动产品创业者
- 互联网个人从业者
- 高等院校和培训机构的师生

代码下载

本书代码下载链接（注意数字与字母大小写）为：

https://pan.baidu.com/s/1c2Ijwpa（密码：kkah）

如果下载有问题，请联系电子邮箱 booksaga@163.com，邮件主题为“Cordova 代码”。

本书作者

本书第 1~15 章由平顶山学院的王亚飞主笔编写，新版本测试由特邀作者王洪飞完成，其他参与人员还有王立平、刘祥淼、彭霁、樊爱宛、张泽娜、曹卉、林江闽、李阳、宋阳、杨超、赵东、李玉莉、刘岩、李雷霆、韩广义等。在此感谢清华大学出版社图格事业部编辑们的辛苦工作，使本书尽早与读者见面。

作者

2017 年 6 月

目　录

第一篇　入门篇

第二篇 基础知识篇



第一篇

入 门 篇

第 1 章

初步了解Cordova

Apache 收购 PhoneGap 后出现了一段时间的平台混乱，很多人不知道是选择 PhoneGap，还是其他项目，但随着 Cordova 的兴起，越来越多人坚定地选择了 Cordova。其实 Cordova 是贡献给 Apache 后的开源项目，是从 PhoneGap 中抽出的核心代码，是驱动 PhoneGap 的核心引擎。

本章的主要知识点有：

- 什么是 Cordova。
- Cordova 的特色。
- 与 Cordova 配套使用的一些 UI 框架。

1.1 认识 Cordova

Cordova 的全称是 Apache Cordova，是一款开放源代码的移动开发框架，原名 PhoneGap。

1.1.1 Cordova 的发展

Cordova 起源于 PhoneGap，那先看看 PhoneGap 的发展。

2008 年 8 月，世界上第一段 PhoneGap 代码诞生了，出现的原因是一名 iOS 程序员无法忍耐 Object-C 生硬而又陌生的语法。而这名程序员又恰恰注意到了 Web 脚本伟大的前景，他发现 Object-C 显然不如简单的 HTML+JavaScript 容易理解，而相对于熟练的 Object-C 程序员，显然熟练的前端开发者更容易找到也更容易培训。于是他就认识到世界上需要这样一种中间件，让 Web 开发者所熟悉的 HTML、CSS、JavaScript 技术能够简单地部署在移动设备上，并且能够同 iPhone 实现简单的功能交互（比如摄像头和重力感应）。

于是伟大的 PhoneGap 就诞生了，图 1-1 为 PhoneGap 的 Logo。也许 Object-C 实在是太遭人厌恶了吧， PhoneGap 一经发布，就已经在 iOS 开发者中间流行起来，获得了许多奖项。PhoneGap 并没有停止前进的脚步，而是将目标瞄准了 Android，并发布了可以支持 Android 平台的框架。这使得 PhoneGap 对移动开发人员来说变得越来越重要。



图 1-1　PhoneGap 的 Logo

2011 年，Adobe 正式宣布收购 PhoneGap，并命名为 Apache Callback。Callback 1.4 版后，更名为 Apache Cordova。Cordova 横跨 Android、iOS、BlackBerry、WebOS、Windows Phone 等主流平台，是目前较强大的一次部署全平台通用的移动开发框架。图 1-2 生动地描述了 Cordova 的跨平台特性。

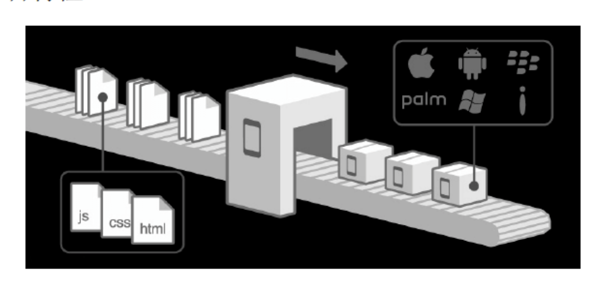


图 1-2　Cordova 的跨平台特性

Apache Cordova 的官方网站是 http://cordova.apache.org/，如图 1-3 所示。可以在这里下载软件或查看文档。如果英文不好的读者，也可以访问中文主页 http://www.cordova.org.cn/。

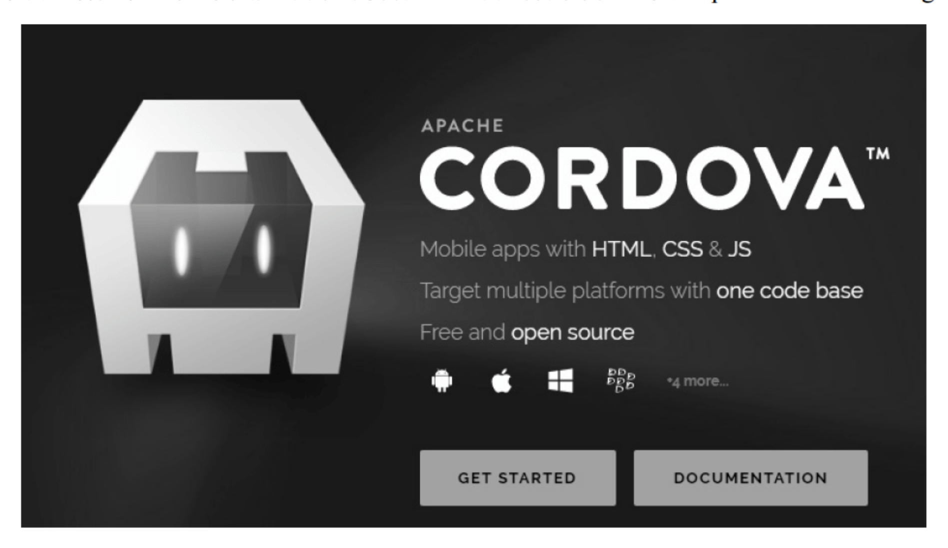


图 1-3　Cordova 官网

1.1.2　Cordova 的特色

在 Cordova 中文主页上，有几行文字概括了 Cordova 的几大特色，如图 1-4 所示。



图 1-4　Cordova 的特色

下面笔者将分条解析它们的精髓。

1. 概述

Cordova 是一款让开发者用普通 Web 技术编写出能轻松调用 API 接口和进入应用商店的 HTML 5 应用开发平台，是一个支持全平台的开源移动框架。Cordova 开发成本低，据估算，其成本顶多为原生 APP 的 1/5。

2. 兼容性

Cordova 完全做到了 Write Once，Run Everywhere，也就是开发者常说的“一次部署，多平台运行”。Cordova 支持常见主流平台的开源移动框架。

Cordova 目前已经可以支持 iOS、Android、BlackBerry、Windows Phone 以及 Web OS 这些主流操作系统。

3. 标准化和 HTML 5+JavaScript

笔者认为这两点如果合在一起会更加贴切，因为标准化中提出的采用 W3C 标准其实就是指采用了标准 HTML 5 进行开发。Cordova 使用将这两点分开来进行强调是出于以下两点目的：

（1）严格来说，W3C 标准是一系列标准的集合，包括但不限于 HTML 5、CSS 3 和 JavaScript。除了这些之外，W3C 标准还包括了一套完整的结构、表现、行为以及命名形式等。将这两点区分开来体现出了 Cordova 开发团队的严谨。

（2）从另一个角度来看，也可能是 Cordova 开发团队利用了 W3C 标准与 HTML 标准界限模糊的“漏洞”，来多次强调同一个问题以制造噱头，图 1-5 为 HTML 5 所涵盖的范围。

从图中不难看出，HTML 5 标准在广义上也完全可以与 W3C 标准处于同一等级上，它也包含了一整套结构、表现、行为以及命名形式的规定。

4. 大众化移动互联网开发平台

目前许多网站在对 Cordova 进行介绍时，总会将主要注意力放在上面提到的三点上，而往往忽略了这最后一点，但是笔者认为这一条才是 Cordova 最为精髓的核心价值。因为很多开发者难以忍受一遍一遍地调试才选择了 Cordova，因此特别看重 Cordova 开发应用时所带来的高效和便捷。

据笔者估计，在实际操作时甚至不需要 20%的开发周期就可以完成所预期的目标，维护成本可能要略大一些，但是也不会超过 20%。笔者经常可以看到身边的一些做营销的朋友，利用 Cordova 在很短的时间内就会开发出一些极其简单的轻应用，然后上传到第三方平台来

获取网店的收益，如果使用原生 SDK 无疑会让人疲于奔命。

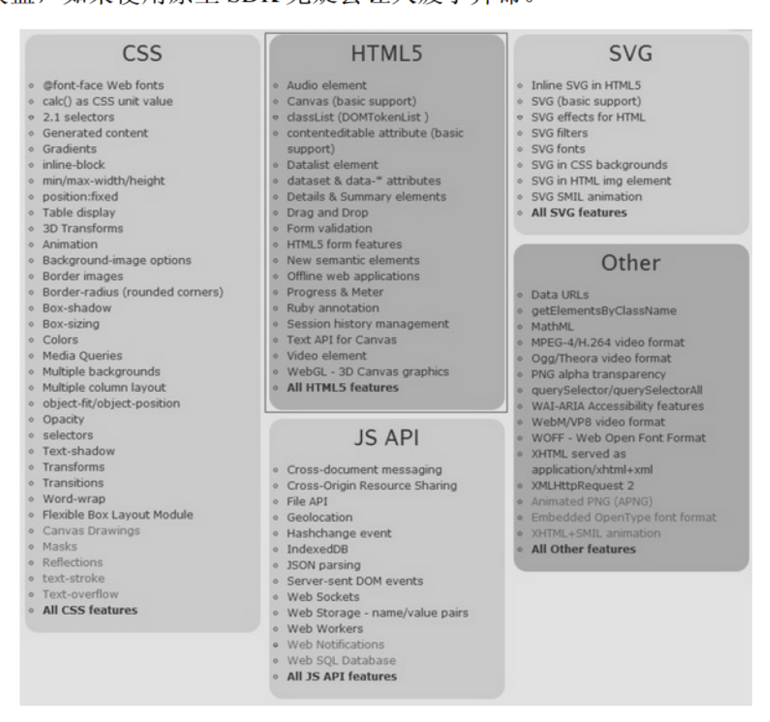


图 1-5　HTML 5 所涵盖的范围



由于使用 HTML 5 只适合开发一些轻量级的应用，如新闻浏览、视频播放或一些棋牌类的小游戏，所以如果有人想要开发一款像“极品飞车”这样的大作，使用 Cordova 倒也不是不可以，毕竟 WebGL 让这一切已经成为理论上的可能，但是他们可能会因为开发成本过高或学习成本过高而被放弃。

虽然 Cordova 开发的便捷性是有局限的，因为它毕竟是一款“轻量级架构”的快速手机开发平台，但是只要它真实有效地提高了开发者的开发效率，那么它就是好样的。

1.1.3　Cordova 的优势

在电影《功夫》中“火云邪神”曾经说过一句话，“天下武功唯快不破”。事实上，不但练武如此，做技术同样如此，尤其是对于许多个人开发者来说，开发应用无非就是为了赚取更多的广告流量或是网店的销售量。对于目前激烈的行业竞争，没有更好的点子恐怕很难在许多同类型的应用中脱颖而出。

既然质量上没有优势，就只能在数量上做一些文章。试想如果别人上传一个 APP 时，有人能够上传 100 个 APP，这无疑就占据了压倒性的优势，可是这也需要有别人 100 倍的开发效率。花大价钱雇人来开发明显不够划算，那就只有想办法提高自己的效率了。

Cordova 显然很适合这种需求，它能够让开发者在极短的时间内开发出功能齐全的应

用，另外还有它跨平台的特性，无疑又一次提高了效率，因此可以说，Cordova 简直就是为个人开发者而生。



Cordova 同样非常适合于团队去操作，只不过优势没那么明显而已，但是仍然值得一试。

1.2 Cordova 的小伙伴们

俗话说“一个好汉三个帮”，再好的框架也不可能应付全部的需求，在许多时候，仅靠着 Cordova 也是不够的，这时就需要一些来自外界的帮助了。

Cordova 一个非常强大的地方在于它能够使用 JavaScript 去操作 API，但是 Cordova 的一个软肋在于它缺少配套的 UI 支持库，在 HTML 中所需要的一切样式都需要开发者利用 CSS 去实现。这样就造成了 Cordova 的三个缺点：

（1）Cordova 的开发者许多是从 Web 转行来的，但是他们未必有移动 Web 的开发经验，在对不同规格屏幕进行适配时往往会出现一些问题，这就给开发者造成很大的困扰。

（2）需要不时去考虑整个应用各个界面间的协调性。

（3）手写 CSS 相比当前多数 SDK 中已经实现了的以拖曳方式设计界面的方法明显不够快捷。

但是，这些对 Cordova 都不能构成实际的威胁，因为毕竟 Web 发展了这么多年，自然有太多“投机取巧”的方法可以借鉴，有太多方便的前端开发框架和工具可以使用。其中比较有代表性的就是 jQuery Mobile、jQuery Touch 和 jQ iPhone UI。

1.2.1 jQuery Mobile

这是最常用的一款基于 HTML 5 的跨平台开发框架，上手非常简单，而且官方给出的范例都非常详细，基本上每一个属性都给出了具体使用的例子。从图 1-6 中可以清楚地看到，官方甚至对每一个版本的功能都进行了分类，可以说非常方便而且人性化。

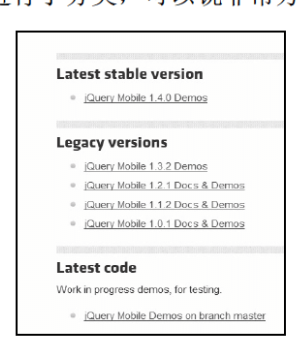


图 1-6 jQuery Mobile 官方对 API 的分类

图 1-7 中是 jQuery Mobile 给出例子的方式，其中包括了对使用方法的解释、实现后的效果和具体的代码。



图 1-7　jQuery Mobile 给出的例子



jQuery Mobile 的说明文档非常人性化，甚至对 jQuery Mobile 没什么了解的人，靠着文档也能摸索着做出自己需要的效果，但前提是要有耐心去读英文才行。

在使用 jQuery Mobile 进行界面设计时，可以采用一款名叫 jQuery Mobile UI Builder 的软件用拖曳的方式来更加方便地生成页面。

另外，关于 jQuery Mobile 还有非常重要的一点，就是 jQuery Mobile 实际上是 jQuery 团队对 jQuery Mobile 在移动 Web 上的一个补充，所以它比其他框架有更多的优势。

1.2.2　jQuery Touch

如果说 jQuery Mobile 作为 jQuery 在移动 Web 的延伸而得以直接使用 Mobile 来命名，那么 jQuery Touch 的命名就完全得益于它强大的事件响应机制了。

与 jQuery Mobile 相比，在构建传统的 Web 应用（如论坛、新闻浏览）时，jQuery Touch 不具有任何优势，但当应用中需要处理一些图标、手势时，jQuery Touch 的优势是显而易见的。表 1-1 是 jQuery Mobile 和 jQuery Touch 的对比。

表 1-1　jQuery Mobile 和 jQuery Touch 的对比

	jQuery Mobile	jQuery Touch
本质	是一个依赖于 HTML 5 的 DOM 操作和 JavaScript、Ajax 操作的 UI 库	这是一个独立的库，包含了自己的 DOM 操作、UI 部件以及全面的事件响应函数
编写	基于 HTML 5，编写简单，与 HTML 完全一致。可方便重用已有的 HTML 代码	以 JavaScript 为核心，重点在于事件的触发和响应，高度依赖于自身
学习难度	非常简单	具有强大的对象模型，学习周期较长，熟练掌握难度较大，但总体上要易于使用原生 SDK
扩展性	易于与其他框架联合使用，扩展性好	扩展性较好但是要弱于 jQuery Mobile
跨平台	跨平台性好，兼容主流浏览器	仅支持 Webkit 内核浏览器

1.2.3　jQ iPhone UI

jQ iPhone UI 提供了一套基于 HTML 5 的 UI 库，似乎是为了更好地证明这一点，它的说明文档也采用了一种独特的方式，如图 1-8 所示。



图 1-8　jQ iPhone UI 特色

由于仅仅是一组 UI 支持库，所以在与 jQuery Mobile 交锋时 jQiPhone UI 几乎没有任何优势。但这也不是绝对的，由于它高度仿真了 iPhone 的原生 UI，因此开发者可以用它来达到一些特殊的目的，比如说构建具有 iOS 风格的 Web 应用或是在安卓系统上模拟 iOS 的界面。

1.3 小结

本章主要介绍了什么是 Cordova，以及 Cordova 的优点。通过本章的学习，读者应该能够认识到 Cordova 是否是自己所需要的“工具”，以及该怎样确保这套“工具”的高效性，同时也应该对学习和使用 Cordova 需要掌握的基础知识（HTML、CSS、JavaScript）有心理准备。

第 2 章

Cordova入门

上一章主要介绍了什么是 Cordova，以及为什么要使用 Cordova。那么本章就来学习如何动手搭建 Cordova 的开发环境。除此之外，本章还将介绍一些相关的知识点，比如 HTML 5、CSS 3、jQuery 等，以及一些在 Cordova 开发中非常有用的小技巧，并在最后开发一个简单的应用。

本章的主要知识点有：

- 安卓开发环境的搭建。
- iOS 开发环境的搭建。
- Cordova 的配置和使用。
- 利用 Node.js 快速部署 Cordova 开发环境。
- 一些在学习 Cordova 时必不可少的基础知识，如 HTML 5、CSS 3、jQuery 等。

2.1 开发环境的搭建

无论是进行哪种语言的开发，最先要做的都是配置相应的开发环境，Cordova 自然也不例外。本节将会以安卓为例介绍 Cordova 开发环境搭建的方法，在其他平台上的搭建方法也与此类似。

2.1.1 安卓开发环境的搭建

在搭建安卓开发环境之前，首先要做的是安装和配置 JDK（Java Development Kit），它是 SUN 公司为 Java 开发者提供的编译工具，可以在百度搜索 JDK 之后到 Oracle 官网（http://www.oracle.com/technetwork/cn/java/javase/downloads/jdk8-downloads-2133151-zhs.html）下载，如图 2-1 所示。



Java SE Development Kit 8u111

您必须接受针对 Java SE 的 Oracle 二进制代码许可协议才能下载该软件。

○ 接受许可协议　◉ 不接受许可协议

产品 / 文件说明	文件大小	下载
Linux ARM 32 硬浮点 ABI	77.78 MB	jdk-8u111-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 硬浮点 ABI	74.73 MB	jdk-8u111-linux-arm64-vfp-hflt.tar.gz
Linux x86	160.35 MB	jdk-8u111-linux-i586.rpm
Linux x86	175.04 MB	jdk-8u111-linux-i586.tar.gz
Linux x64	158.35 MB	jdk-8u111-linux-x64.rpm
Linux x64	173.04 MB	jdk-8u111-linux-x64.tar.gz
Mac OS X	227.39 MB	jdk-8u111-macosx-x64.dmg
Solaris SPARC 64 位	131.92 MB	jdk-8u111-solaris-sparcv9.tar.Z
Solaris SPARC 64 位	93.02 MB	jdk-8u111-solaris-sparcv9.tar.gz
Solaris x64	140.38 MB	jdk-8u111-solaris-x64.tar.Z
Solaris x64	96.82 MB	jdk-8u111-solaris-x64.tar.gz
Windows x86	189.22 MB	jdk-8u111-windows-i586.exe
Windows x64	194.64 MB	jdk-8u111-windows-x64.exe

图 2-1　在百度中搜索 JDK 或官网下载

步骤 01　在官网中最下面两个版本分别是 Windows32 位和 64 位，下载前先选中上方的“接受许可协议”，然后点击相应版本右侧的链接，如图 2-2 所示。下载后的文件名和文件大小如图 2-3 所示。



本书默认使用 Windows 操作系统，可以根据需要选择 32 位或者 64 位。

Java SE Development Kit 8u111

您必须接受针对 Java SE 的 Oracle 二进制代码许可协议才能下载该软件。

○ 接受许可协议　◉ 不接受许可协议

产品 / 文件说明	文件大小	下载
Linux ARM 32 硬浮点 ABI	77.78 MB	jdk-8u111-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 硬浮点 ABI	74.73 MB	jdk-8u111-linux-arm64-vfp-hflt.tar.gz
Linux x86	160.35 MB	jdk-8u111-linux-i586.rpm
Linux x86	175.04 MB	jdk-8u111-linux-i586.tar.gz
Linux x64	158.35 MB	jdk-8u111-linux-x64.rpm
Linux x64	173.04 MB	jdk-8u111-linux-x64.tar.gz
Mac OS X	227.39 MB	jdk-8u111-macosx-x64.dmg
Solaris SPARC 64 位	131.92 MB	jdk-8u111-solaris-sparcv9.tar.Z
Solaris SPARC 64 位	93.02 MB	jdk-8u111-solaris-sparcv9.tar.gz
Solaris x64	140.38 MB	jdk-8u111-solaris-x64.tar.Z
Solaris x64	96.82 MB	jdk-8u111-solaris-x64.tar.gz
Windows x86	189.22 MB	jdk-8u111-windows-i586.exe
Windows x64	194.64 MB	jdk-8u111-windows-x64.exe

图 2-2　选择下载 JDK

jdk-8u111-windows-i586.exe	2017/1/9 星期一 ...	应用程序	193,757 KB

图 2-3　下载后 JDK

步骤02 双击运行安装程序，得到如图 2-4 所示的界面，点击“下一步”按钮继续安装。

步骤03 继续点击“下一步”按钮，JDK 开始安装。稍等几分钟后，JDK 安装完毕，得到如图 2-5 所示的界面。接下来可以点击“后续步骤”按钮查看 JDK 配置环境变量的方法。



图 2-4 欢迎界面



图 2-5 JDK 安装完成



实际上，随着新版本的不断出现，在新版本的 JDK 安装完成之后就已经可以在 CMD 中使用 java 命令来调用。但是如果在 CMD 中执行命令 javac 却会显示出提示：javac 不是内部或外部命令，也不是可运行的程序或批处理文件。如果是这样，请继续下面的步骤。

步骤04 在“计算机”图标上点击鼠标右键，在弹出的快捷菜单中选择“属性”命令，打开如图 2-6 所示的界面。选中左侧的“高级系统设置”选项，就弹出如图 2-7 所示的对话框。

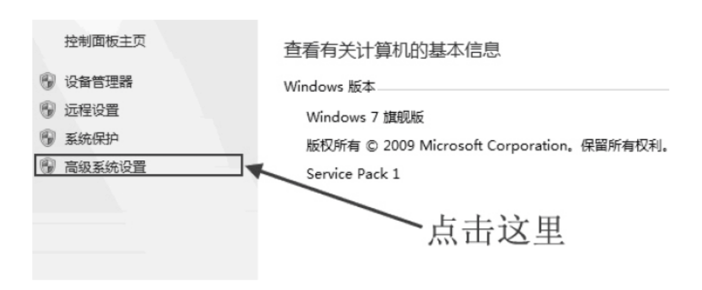


图 2-6 选择“高级系统设置”

步骤05 在某些系统中还要选中“高级”选项卡才能得到图 2-7 所示的界面，进入之后点击底部的“环境变量”按钮就可以对环境变量进行设置了，如图 2-8 所示。

步骤06 新建一个系统变量，命名为 JAVA_HOME，其中的内容为：C:\Program Files\Java\jdk1.7.0_51（一定注意这个后面没有分号）。再新建环境变量 classpath，内容为：.;%java_home%\lib。之后还要在环境变量中找到 path 变量，在其中加入内容：;%java_home%\bin。



图 2-7　点击“环境变量”按钮



图 2-8　在此处对环境变量进行设置

注意：是在原有的内容之后加入而不是替换，如原内容为：

```
C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common;C:\Program Files (x86)\
Intel\iCLS Client\;
%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\;
```

那么修改后的内容应为：

```
C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common;C:\Program Files (x86)\
Intel\iCLS Client\;
%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\;%java_home%\bin。
```



要在原内容的最后先加入一个分号，将内容隔开。

在配置环境变量时要注意，环境变量实际上是 JDK 安装的真实路径，比如说在本例中 JDK 就被安装在了路径 C:\Program Files (x86)\Java\jdk1.8.0_111 下，如图 2-9 所示。



图 2-9　JDK 的安装路径

步骤 07　配置完成后，在 CMD 中输入 javac - version，显示所安装 JDK 的版本号即可证明环境变量配置成功，如图 2-10 所示。

```
C:\Users\Administrator>javac -version
javac 1.8.0_111
```

图 2-10　显示 JDK 版本号

步骤 08　在完成了 JDK 的配置之后就可以搭建安卓的开发环境了。本来这是非常复杂的一项任务，但是现在安卓官方已经为开发者提供了打包好的集成开发环境，只要去官网下载后解压即可。在浏览器中输入网址“http://developer.android.com/”可以到安卓的开发者网站上下载最新版本的 SDK（Software Development Kit，即软件开发工具包），如图 2-11 所示。

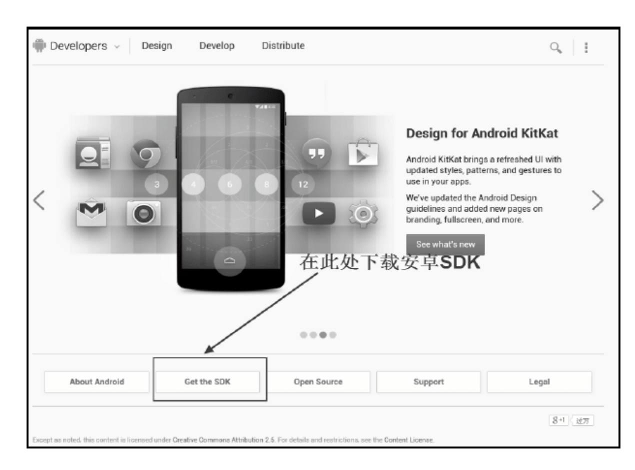


图 2-11　在安卓开发者主页获取 SDK

步骤 09　点击屏幕下方的 Get the SDK 按钮可以进入如图 2-12 所示的页面，点击右侧的 Download the SDK 按钮，选择要下载的 SDK 版本。

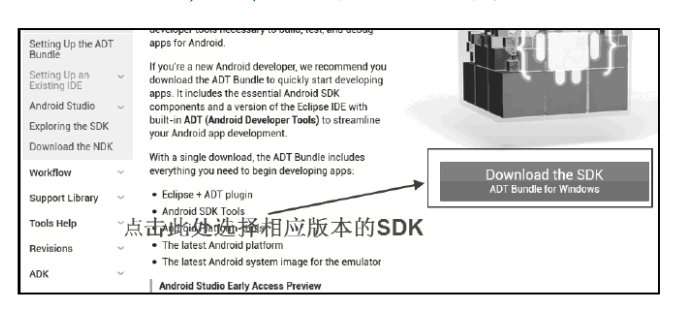


图 2-12　进入 SDK 下载页面



安卓 SDK 又叫做 ADT，是 Android Developer Tools（安卓开发者工具）的缩写。

步骤 10　选中 I have read and agree with the above terms and conditions 复选框，根据需要选择是使用 32 位还是 64 位的开发环境并点击 Download the SDK ADT Bundle for Windows 按钮就可以开始下载了。注意，此处只提供了 Windows 环境下的 SDK，

如果使用 Linux 还需要下载 Eclipse 进行配置。



由于谷歌在我国不时被屏蔽导致读者无法顺利地连接到安卓开发者网站获取 ADT，因此读者可在百度搜索相关下载包。

步骤 11 下载之后将它解压并复制到 D 盘根目录下（也可以是其他位置，但要保证其中不要有中文路径）。打开 D:\adt-bundle-windows-x86_64-20131030\eclipse 路径下的 eclipse 即可进入安卓开发环境（如果提示 javaw 找不到的错误，要修改 eclipse.ini 中 javaw 的路径，具体方法可百度）。

步骤 12 在菜单中选择 File|New|Android Application Project，弹出如图 2-13 所示的窗口，按照图中的内容进行填写并点击 Next 按钮。

步骤 13 之后一直点击 Next 按钮，最终来到如图 2-14 所示的界面，点击 Finish 按钮完成新项目的创建。

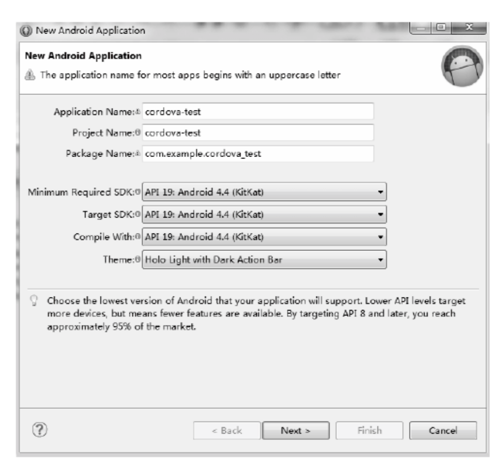


图 2-13 新建一个安卓项目

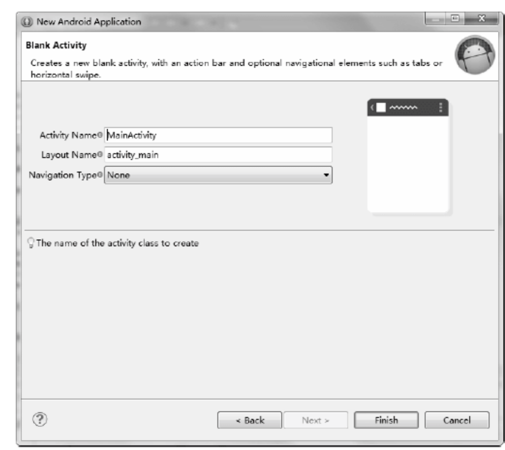


图 2-14 完成新项目的创建

步骤 14 由于安卓已经为开发者准备了简单的例子，因此当新建项目之后，实际上已经是一个简单且完整的例子了，可以直接编译运行。但是为了运行它，还需要新建一个虚拟机才行。在菜单中选中 Window|Android Virtual Device Manager，弹出如图 2-15 所示的窗口。

步骤 15 点击 New 按钮，新建一个虚拟机，其中在 AVD Name 一栏填写虚拟机的名称，可以随意填写，由于笔者对 HTC 发布的 Nexus One 有着极深的热爱，因此常常会以此来作为名称。Device 一栏用来选择合适的设备分辨率以及屏幕尺寸。Target 一栏用来选择所使用安卓系统的版本，一般来说 SDK 中会自带当前最新版本的系统，比如本例中使用的是 4.4 版本。如果想使用老版本则需要自己去下载。此外还可以在下方的其他选项中去设置虚拟机的内存以及 SD 卡等内容。点击 OK 按钮完成创建，如图 2-16 所示。

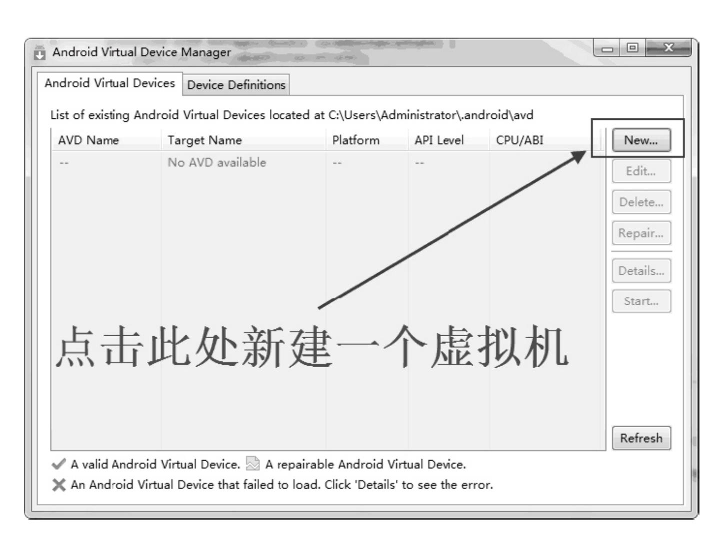


图 2-15　新建一个虚拟机

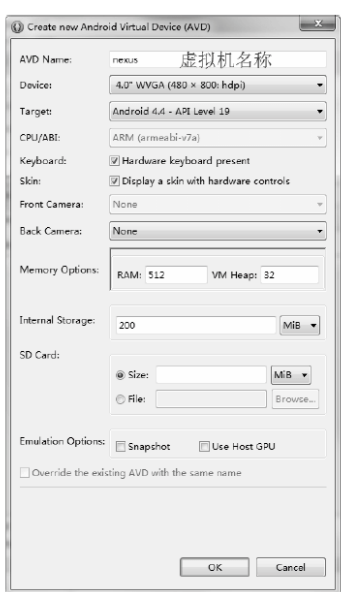


图 2-16　设置新建虚拟机的各项属性

步骤 16　接下来就可以选中刚刚新建好的安卓项目并在其上右击，在弹出的快捷菜单中选择 Run AS|Android Application 进行测试，结果如图 2-17 所示。



在第一次打开虚拟机时，由于系统需要进行安装和加载，因此等待的时间会比较长，请各位读者耐心一些。另外在第一次测试时为了能够减少等待时间，建议读者可以将虚拟机的屏幕分辨率尽量降低一点。

至此，安卓的开发环境就配置成功了，本章的后续内容将逐步介绍一些它的使用技巧和方法，请读者注意实践。



图 2-17　第一个安卓程序

2.1.2　iOS 开发环境的搭建

iOS 开发环境的配置则要容易很多，当然了，如果想要开发 iOS 程序，最好有一台 mac。

步骤 01　打开 mac 的 App Store，找到一款软件 Xcode，如图 2-18 所示。

步骤 02　下载后安装并运行如图 2-19 所示。



图 2-18　iOS 开发工具

图 2-19　Xcode 运行后的界面

步骤 03　如果是要创建一个全新的项目，就选择第二行 Create a new Xcode project，然后打开图 2-20 所示的界面。

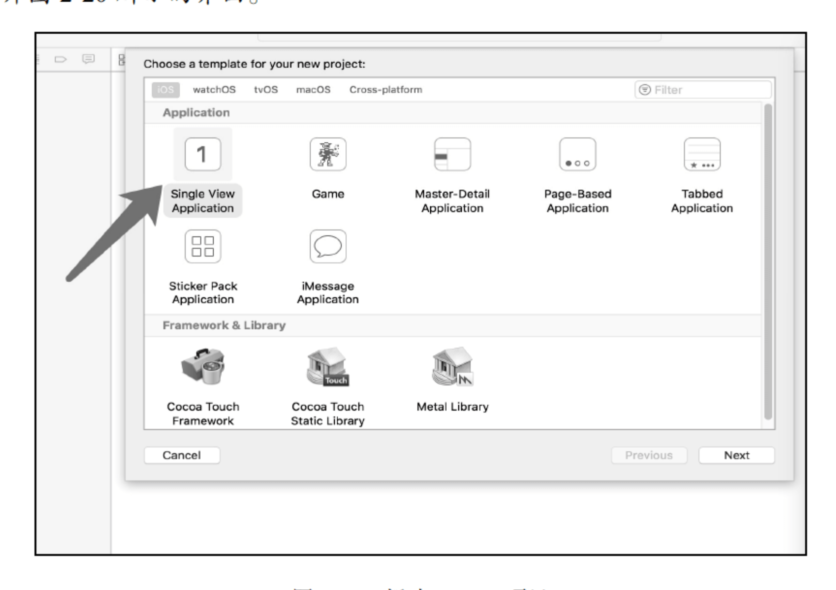


图 2-20　新建 Xcode 项目

步骤 04　如果是要开发一般的手机 App，选择 Single view Application，然后点击 Next 按钮后如图 2-21 所示。

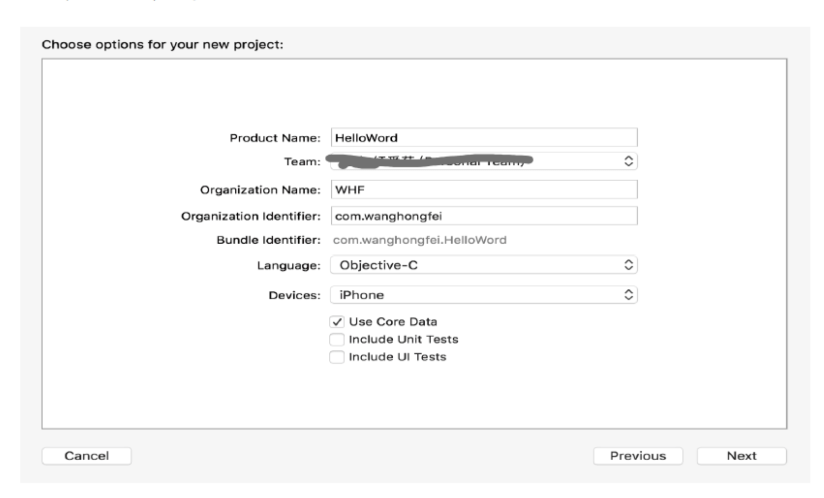


图 2-21　填写 APP 信息

步骤 05　给项目起一个名字，然后点击 Next 按钮，选择一个路径后，第一个程序就算是创建完成了，新建 APP 的工程界面如图 2-22 所示。

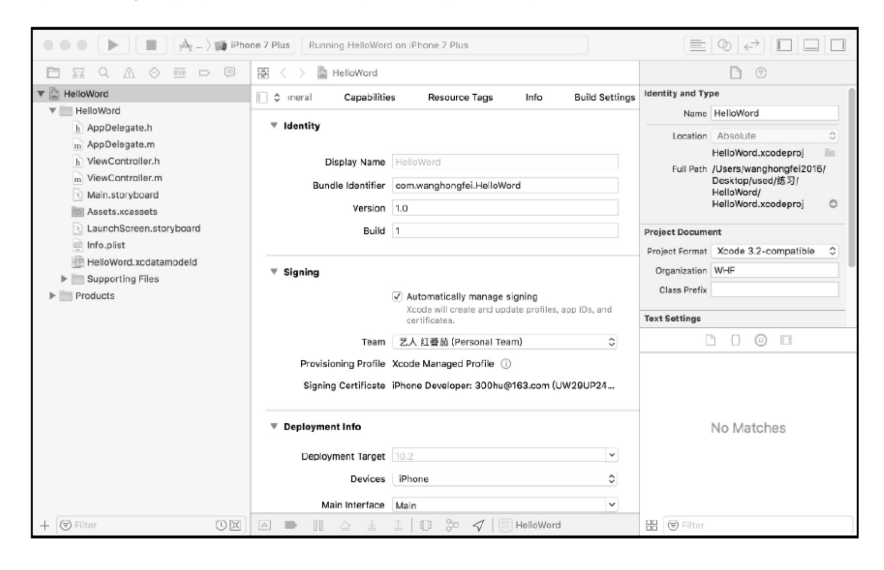


图 2-22　新建项目界面

点击左上角的三角按钮，就可以调用模拟器运行程序了，只不过这个程序什么都没有添加，能看到的只有一个白色背景。因为本书并不是要讲解 iOS 编程，所以就不做过多地介绍了。

另外，因为要在 mac 上面书写 JavaScript 和 HTML 的代码，建议读者再去下载一个其他的编程工具，只要能编写 JavaScript 和 HTML 的代码即可，个人比较推荐 HBuilder，如图 2-23 所示。下载地址：http://www.dcloud.io/index.html。



图 2-23　HBuilder

2.1.3　Cordova 的配置

为了提高运行速度和降低开发者的工作量，Cordova 在 3.0.0 版本之后采用了全新的架构，利用 Node.js 自动生成项目。目前最新的 Cordova 版本是 6.3.0。

Cordova 使用 Node.js 进行管理。首先要去 Node.js 的官方网站（http://nodejs.org/）下载最新版本的 Node.js 并进行安装（这个安装有 msi 安装版，比较简单，所以本书省略安装步骤）。安装完成后以管理员身份运行 Node.js command prompt.exe。



许多人在这一步容易错误地运行了 Node.js.exe 进行操作，因为它的图标比较容易成为用户的目标，但这却是错误的。

运行 Node.js command prompt.exe 之后，在命令行中输入命令：npm install –g cordova，Node.js 将会自动在网络上寻找 Cordova 相关组件进行安装。稍等几分钟即可下载完毕，可以进行下一步的操作。在命令行下执行命令 cordova create myCordova，创建一个名为 myCordovap 的项目，完成后出现如图 2-24 所示的提示。此时在用户目录下已经出现了一个名为 myCordovap 的文件夹，如图 2-25 所示。双击打开该文件夹，如图 2-26 所示。

```
C:\Users\Administrator>cordova create myCordova
? May Cordova anonymously report usage statistics to improve the tool over time?
You have been opted out of telemetry. To change this, run: cordova telemetry on.

Using detached cordova-create
Creating a new cordova project.
```

图 2-24　用命令行创建一个新项目

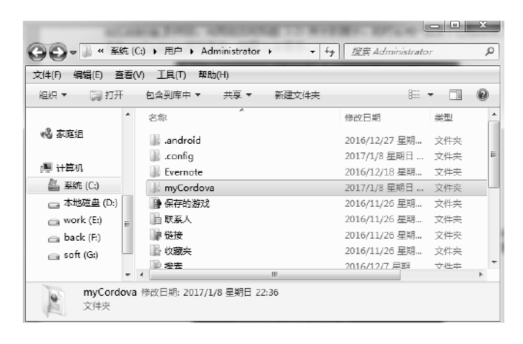




图 2-25　新创建的项目可以在用户目录中找到　　　　图 2-26　新创建的项目结构

在命令行中继续输入命令：cd myCordova，进入项目文件，然后使用 cordova platform

add android 命令，为当前项目添加 Android 平台。如果是 iOS 平台，将 android 替换为 ios 即可。如果第一次添加不成功，记得用 cordova platform remove android 先删除之前添加的 Android 平台。第一次添加平台会比较慢，如图 2-27 所示。

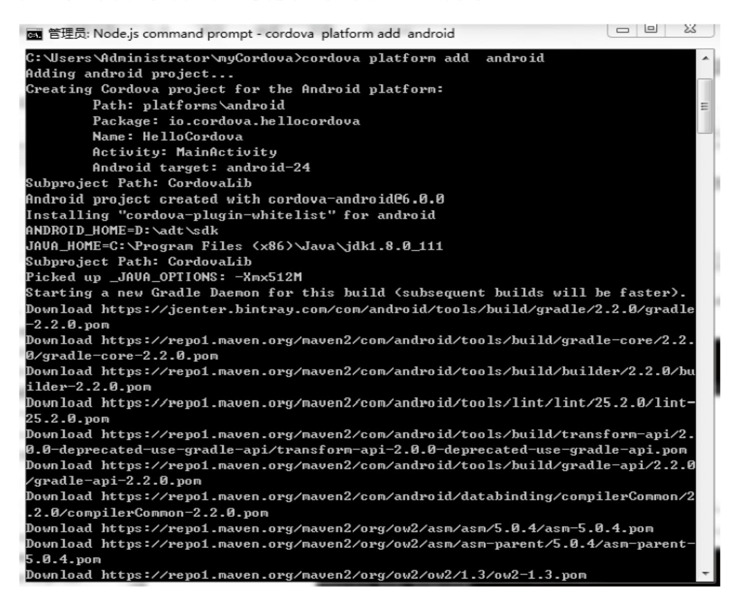


图 2-27　添加安卓平台（第一次会比较慢）

添加平台后，发现在 myCordova\platforms 下多了一个 android 文件夹，此时就可以将整个文件夹导入到 Eclipse 中进行使用了。导入后包含两个项目，一个是 CordovaLib，一个是 MainActivity。MainActivity 为测试项目，CordovaLib 为 Cordova 的源代码。

还可以使用 cordova platform list 来查看当前项目，添加一个平台，如果想都编译这些平台，使用 cordova build 生成可运行的 app 文件，如 Android 平台下的 apk 文件。如果只想生成某个平台，可以用 cordova build android（iOS 开发要运行 cordova build ios）来指定平台进行生成，如图 2-28 所示。

```
C:\Users\Administrator\myCordova>cordova build android
ANDROID_HOME=D:\adt\sdk
JAVA_HOME=C:\Program Files (x86)\Java\jdk1.8.0_111
Subproject Path: CordovaLib
Downloading http://services.gradle.org/distributions/gradle-2.14.1-all.zip
```

图 2-28　利用 Cordova 生成完整的安卓 apk 文件（第一次会比较慢）

Cordova 是命令行界面（简称 CLI），所以一般把它的操作称为 Cordova CLI，具体 CLI 都有哪些命令，可参考官方文档：http://cordova.apache.org/docs/en/latest/guide/cli/index.html。或者也可以通过 cordova help 命令来查看。



还可以在网站 https://build.phonegap.com/上利用 Cordova\PhoneGap 提供的在线编译工具直接生成各个平台都能使用的安装文件，这样就彻底抛开了 SDK，真正做到了跨平台开发，非常方便。

2.2 跨平台的 HTML 5

Cordova 是一套基于 HTML 5 的跨平台开发框架，要学习使用 Cordova，首先要熟练掌握 Cordova 的 API，归根到底还是要对 HTML 5 使用熟练。目前，开发者对 HTML 5 的认识还存在着不小的误区。因此掌握好 HTML 5 的基本知识是非常重要的。

HTML 5 是用来取代在 1999 年被指定并沿用至今的 HTML 4 和 XHTML 标准的。虽然它目前仍处于不断完善阶段，但却已经不断地渗透到了人们的日常网络生活当中，大多数的浏览器目前都能够支持 HTML 5。

相对于过去的 HTML 标准，HTML 5 主要有以下两大改进：一是强化了网页的表现效果，包括了 Canvas 动画（如图 2-29 所示）、对三维效果和 CSS 3 的支持，以及对摄像头等设备的支持等；二是追加了本地数据库等 Web 应用的功能。



图 2-29 利用 Canvas 动画制作的 HTML 5 游戏

功能的强大使 HTML 5 已经不再是单纯的通用标记语言了，现在提到 HTML 5 时往往指的是包含了 HTML 5、CSS 3、jQuery 甚至是设计模式在内的一系列技术组合。它能够有效地减少浏览器对插件（如 Adobe Flash 等）的依赖，并可提供更加强大的服务。图 1-5 为 HTML 5 所包含的技术范围。



减少对插件的依赖就代表着有更多的应用可以仅依靠浏览器而独立运行，不必担心平台的限制。这就导致了 HTML 5 强大的跨平台特性，使得同一款应用可以在不同的设备上都能够运行。这也是为什么目前主要的跨平台开发框架大多是基于 HTML 5 的原因。除此之外，也不难解释当年 iOS 敢于“不支持”Flash 的原因了。

广义来说，只要在 HTML 文件的头部加入一句声明“<!DOCTYPE html>”就证明该页面是支持 HTML 5 标准的了。这也就导致了目前许多没有良心的商人利用文字上的歧义，来宣称自己的产品是基于 HTML 5 的，以此骗取高额的收益。

笔者在这里希望本书的读者都能够具备明辨是非的能力，不要被这种黑心的商人所蒙骗，那么 HTML 5 究竟包含了哪些新的技术呢？下面将一一列举出来。

（1）语义特性：HTML 5 包含了更加丰富的标签，包括<header>、<menu>等，能够使网页的结构具有更好的可读性，便于搜索引擎的理解和收录。这些标签也使得页面具有了更好的“模块化”特性，便于后期的维护和修改。

（2）本地存储特性：为 HTML 5 提供了本地缓存功能，使其能够存储一些本地数据让基于 HTML 5 的 APP 能更快地启动和加载。

（3）设备兼容特性：由于 HTML 5 致力于减少对外部插件的依赖，并提供了前所未有的数据接入接口，这使得 HTML 5 具有了比前一代 HTML 更加强大的设备兼容性。

（4）连接特性：加强了对 Ajax 的支持，这使得 HTML 5 能够提供更好的网页实时聊天、在线网页游戏等服务。其中两个重要的特性分别是 Sever-Sent-Event 和 WebSocks，这两个特性能够更加有效地将来自服务器端的数据实时推送至客户。

（5）三维以及图形特效特性：支持基于 SVG、Canvas、WebGL 以及 CSS 3 的动画功能，使 HTML 5 具有了更强大的图形处理能力，理论上已经能够支持一些大型 3D 游戏。

（6）网页多媒体特性：包括<video>、<audio>标签，以及对摄像头等外设的支持，使 HTML 5 具有了强大的多媒体处理能力。

（7）性能优化：使浏览器能够更加快速地实现对页面的渲染，保证用户不需要长时间等待页面的加载。



HTML 5 非常强大，但并不代表它很安全，反而由于它的诸多新特性给开发者和维护管理人员带来了更多的挑战。使用 HTML 5 编写的应用甚至会比传统应用更容易泄露用户的敏感数据。欧洲网络信息安全机构（ENSIA）已经警告说 HTML 5 不够安全。

2.3 更好玩的 CSS 3

CSS 是层叠样式表（Cascading Style Sheet）的缩写，在网页中使用 HTML 可以对页面元素的颜色、字体、背景和其他效果进行有效且精确地控制。

CSS 3 是在 HTML 5 标准中根据需要针对过去 CSS 进行的修改和补充，主要包括以下内容。

- 盒子模型的样式：主要包括了圆角、阴影、透明以及背景渐变等新的效果。
- 选择器：为 CSS 3 提供了比旧版本的 CSS 更加灵活的选择器模型。
- 文字特效：使 CSS 3 支持颜色渐变、阴影文字多列显示等效果。
- 2D/3D 转换：为 CSS 动画奠定基础，使页面元素支持更广泛的几何变换。
- 动画：为 CSS 加入了新的动画特性，使 HTML 5 代替 Flash 成为可能。

通过使用 CSS 3 能够完成许多在原有 CSS 框架下看似不可能的任务，比如说可以用 6 个 div 元素与 CSS 3 混用来实现正方体的效果。

首先在 HTML 中建立 6 个元素，为了使它们便于定位，要使用两层 div 对它们进行嵌套。分别用 1~6 的阿拉伯数字来标注正方体的 6 个面：

```
<section class="container">
   <div id="cube">
      <figure class="front">1</figure>
```

```
        <figure class="back">2</figure>
        <figure class="right">3</figure>
        <figure class="left">4</figure>
        <figure class="top">5</figure>
        <figure class="bottom">6</figure>
    </div>
</section>
```

接下来利用 CSS 定义各个层次的关系、尺寸，为进行 3D 变换做准备。

```
.container {
  width: 200px;
  height: 200px;
  position: relative;
  <span style="color:#ff0000;">perspective: 1000px;</span>
}
#cube {
  width: 100%;
  height: 100%;
  position: absolute;
 <span style="color:#ff0000;"> transform-style: preserve-3d;</span>
}
#cube figure {
  width: 196px;
  height: 196px;
  display: block;
  position: absolute;
  border: 2px solid black;
}
```

接下来要为正方体的 6 个面做 3D 变换，由于每个面的面积大小都是完全一样的，不同的只是所处的坐标和角度，因此在此部分只需要利用 translation 属性进行变换，比如说 font 需要旋转 180°，然后向页面内部平移 100px：

```
#cube .front  { transform: rotateY(   0deg ) translateZ( 100px ); }
#cube .back   { transform: rotateX( 180deg ) translateZ( 100px ); }
#cube .right  { transform: rotateY(  90deg ) translateZ( 100px ); }
#cube .left   { transform: rotateY( -90deg ) translateZ( 100px ); }
#cube .top    { transform: rotateX(  90deg ) translateZ( 100px ); }
#cube .bottom { transform: rotateX( -90deg ) translateZ( 100px ); }
#cube.show-front  { transform: translateZ( -100px ) rotateY(    0deg ); }
#cube.show-back   { transform: translateZ( -100px ) rotateX( -180deg ); }
#cube.show-right  { transform: translateZ( -100px ) rotateY(  -90deg ); }
```

```
#cube.show-left   { transform: translateZ( -100px ) rotateY(   90deg ); }
#cube.show-top    { transform: translateZ( -100px ) rotateX( -90deg ); }
#cube.show-bottom { transform: translateZ( -100px ) rotateX(   90deg ); }
```

最后为正方体加入动画效果使正方体不断滚动：

```
#cube { transition: transform 1s; }
```

最终实现的效果如图 2-30 所示。

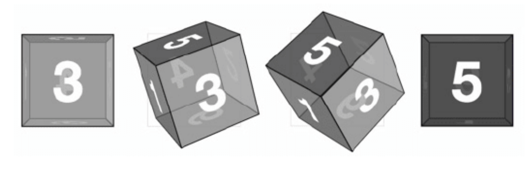


图 2-30　利用 CSS 3 所实现的正方体效果

总结：CSS 3 能够完成一些在旧版本 CSS 中无法完成的任务，它在未来的开发中会起到举足轻重的作用。



在实际开发中，可以利用本例来实现游戏中摇骰子的效果，以增强用户的娱乐性。另外在日常的应用中，动画也是增强交互必不可少的要素，比如 360 手机助手中就为“加速”功能加入了一个火箭升空的动画。虽然这些动画效果用 JavaScript 也可以实现，但是效率远不如 CSS 3。另外，网页游戏界也有一条规则是：“能用 CSS 3 动画就别用 JavaScript”。由此不难看出 CSS 3 的重要。

2.4 完美兼容浏览器的 jQuery 框架

jQuery 是一个优秀的轻量级 JavaScript 框架，能够完美兼容 HTML 5、CSS 3 以及各种浏览器，并使开发者能方便地处理 HTML 事件、动画，也能为网站提供方便的 Ajax 交互。

相对于其他 JavaScript 框架，jQuery 一个非常大的优势在于它拥有非常全面的帮助文档和例程，并且拥有许多成熟的插件可供选用，比如 jQuery Mobile 就是一套非常不错的移动插件。此外，jQuery 不但能够有效地使 jQuery 代码与 HTML 内容分离，也可以与原本的 JavaScript 混用，这对开发者来说无疑是非常方便的。

比如，要使用传统的 JavaScript 响应页面上某元素被点击的事件，需要构建如下代码：

```
<script>
Function do() {
    // 此处填入相应的操作
}
</script>
<div onclick="do();"></div>
```

而如果在 jQuery 中，则只需要编写如下代码：

```
$(document).ready( function() {
    $("#button").onclick( function() {
        // 此处填入相应的操作
   });
  });
```

虽然代码看似变长了，也更复杂了，但这却完美地实现了 HTML 与 JavaScript 代码的分离，因此在实际开发中是非常方便的。当然，在一些简单的小项目中，有些开发者并不习惯 jQuery 这种语法，那么还可以选择这样：

```
function do() {
    $(div).style.left = "100px";                    // 混用 jQuery 与原生 
JavaScript
}
```

另外，本书介绍的是使用 HTML 5 进行手机应用的开发，许多插件都是基于 jQuery 的，这也是许多开发者不得不对 jQuery 有所了解的一个重要原因，图 2-31 是 jQuery Mobile 的一个界面。

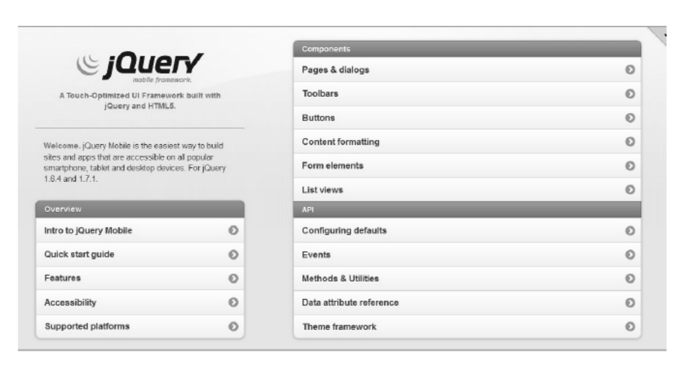


图 2-31　jQuery Mobile 制作的界面

2.5 小结

目前，几乎所有开发者都热衷于开发独立的移动应用，因为这会为他们带来巨大的收益。但是在不久的将来，这些一定会被 HTML 5 甚至是 Cordova 终结。因为无论是 Cordova 还是 HTML 5，都在致力于完成同一个目的：将更加强大的功能进行封装使之实现跨平台的特性。Cordova 在本质上就是编译一个更加强大的跨平台浏览器，而这有可能会将当前的软件开发行业带入一个新的时代。

第 3 章

开始前的准备

本章将在正式学习 Cordova 之前先做一些必要的准备工作，以便更好地体验 Cordova 所带来的便利性和高效性。此外，本章还将对 HTML 5 的概念和一些功能进行更加深入地学习，以便使读者能够更好地适应未来发展的趋势。本章的最后将对 Cordova 中的各种 API 做一个简单的总结。

本章的主要知识点有：

- 与 HTML 5 相关的一些知识。
- 使用 Cordova 配合 jQuery Mobile 快速开发简单应用的方法。
- 开发 Cordova 应用时，为了保证应用的适应性而要注意的一些细节。
- Cordova 中各类 API 的应用场合。

3.1 HTML 5，你真的准备好了吗

其实 Cordova 的开发归根到底还是 HTML 5 的开发，作为一种正在不断成长中的新技术，HTML 5 的发展无疑为开发者提供了很好的机会。在进行学习之前笔者整理了在 HTML 5 面试时最常被问到的 10 道面试题以及答案，读者可以以此来测试自己是否真的准备好了。

1. HTML 5 中新的 DocType 和 Charset 作用是什么

与 HTML 4 相比，HTML 5 已经不再是 SGML（标准通用标注语言）的子集，因此可以用简化的方式对文档进行声明。具体方法如下：

```
<!doctype html>                         <!--HTML 5中对 doctype 的定义-->
<meta charset="UTF-8">                 <!--HTML 5中对字符编码形式的定义-->
```

2. 如何在 HTML 5 页面中嵌入音频

HTML 5 包含了嵌入音频的标准方式，支持的格式包括 MP3、Ogg 和 Wav。具体使用方式的例子如下：

```
<audio controls>
```

```
  <source src="jamshed.mp3" type="audio/mpeg">
   你的浏览器不支持音频
</audio>
```



在 W3C 上有关于各种浏览器对不同音频文件格式兼容性的说明，但是那个说明还是在 Firefox 4.0 时代发布的；而目前 Firefox 已经发布了 50.0 版本，有许多支持特性已经大大增强，因此在使用这些功能时建议还是实际试一试。

3. 如何在 HTML 5 页面中嵌入视频

和音频一样，HTML 5 定义了嵌入视频的标准方法，支持的格式包括：MP4、WebM 和 Ogg，使用方法的例子如下：

```
<video width="450" height="340" controls>
<source src="jamshed.mp4" type="video/mp4">
        你的浏览器不支持视频
</video>
```



在传统互联网视频点播系统（也包括目前）中，使用 Flash 的方式仍然占据了重要地位，因此要让 HTML 5 的视频功能真正发展起来还有一段路要走。

4. 除了音频和视频，HTML 5 还支持哪些新的媒体元素

除了 video 和 audio 之外，HTML 5 中还支持 embed（作为外部应用的容器）、track（定义对媒体的文本跟踪）和 source（用于增强对多种媒体源的支持）三种媒体元素。

此外，被广大 HTML 5 爱好者喜闻乐见的 Canvas 元素也可以认为是媒体元素的一种。

5. HTML 5 中的 Canvas 元素应该怎样使用

Canvas 元素是在 HTML 5 中利用脚本（主要是 JavaScript）来实现在页面中进行绘图操作的元素。

这个元素充其量只是图像的一个容器，起到画布的作用，而一切操作（绘图或者动画）都是通过额外的脚本来完成的。

6. HTML 5 有哪些不同类型的存储

与以往 Web 应用通过 Cookie 实现存储功能不同，HTML 5 中新加入了本地存储功能，可以实现快速而又安全的本地数据存储服务。

HTML 5 提供了两种不同的对象可用来存储数据：

- localStorage：适用于长期存储数据，浏览器关闭后数据不丢失。
- sessionStorage：存储的数据在浏览器关闭后自动删除。

7. HTML 5 引入了什么新的表单属性

HTML 5 中引入了大量新的表单属性，如表 3-1 所示。

表 3-1　HTML 5 中引入的表单属性

属性名称	说明
datalist	与 input 标签联合使用，用于声明 input 标签中的各个选项
datatime	用于确定 input 标签中输入内容的类型
output	用于定义不同类型的输出
keygen	规定用于表单生成器的密钥
date	用于规定 input 标签中输入的类型，必须为一个日期
month	用于规定 input 标签中输入的类型，为一个数字
week	用于规定 input 标签中输入的类型
time	用于规定 input 标签中输入的类型
number	用于规定 input 标签中输入的类型
range	用于规定 input 标签中输入的类型，表示 input 类型为滑动条
email	用于规定 input 标签中输入的类型
url	用于规定 input 标签中输入的类型

8. 与 HTML 4 相比，HTML 5 废弃了哪些元素

出于各种原因，一些旧的元素在 HTML 5 中将不再被使用。比如说元素 basefont、big、center、font、strike 和 tt 由于能够被 CSS 很好地代替而不再使用，而一些其他元素如 frame、frameset 和 noframes 则因为可能会破坏页面的可用性和可访问性而被废弃，而元素 acronym、applet、isindex 和 dir 则是被 HTML 5 中的新元素取代。

另外，还有一大批属性也一同在 HTML 5 中被废除。



由于开发者已经熟悉了那些“老的”元素以及属性，因此在现实中 HTML 5 与 HTML 4 混用的情况经常发生。这会给未来的维护工作带来不少的麻烦，因此作为开发者一定要努力去避免这种情况发生。

9. HTML 5 标准提供了哪些新的 API

HTML 5 提供了多种新的 API 以支持它的诸多新特性，如 Media（媒体）、Text Track（文本跟踪）、Application Cache（应用缓存）、User Interface（用户接口）、History（历史记录）等。

10. HTML 5 应用缓存和常规浏览器缓存有何差别

HTML 5 的应用缓存最关键的就是支持离线应用，可获取少数或者全部网站内容，包括HTML、CSS、图像和 JavaScript 脚本并存在本地。该特性加速了网站的性能，与传统的浏览器缓存比较，该特性并不强制要求用户访问网站。

一般来说，在上述 10 个问题中，能够答对 7 个以上就可以算是对 HTML 5 有了一定的了解，而答对 9 个以上的读者恭喜你！你已经在未来的 HTML 5 热潮中占据了先机。

3.2 HTML 5 的若干练习

上一节提出了 10 个关于 HTML 5 的问题，如果你能够答对 7 个以上，那你可以直接跳过本节了。但是如果你没有答对 6 个问题，还请你耐心地在本节中接受几个实战锻炼，跟着笔者用 HTML 5 来实现几个简单的界面。

3.2.1　实现渐变的背景和圆角的按钮

在 Cordova 中，一切的界面都是利用 HTML 5 来实现的，其中背景颜色和按钮是非常重要的两个元素，那么在本小节中将开始实现一组利用 HTML 5 制作的按钮样式。

【范例 3-1】利用 HTML 5 实现渐变的背景和圆角的按钮。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>利用 HTML 5实现渐变的背景和圆角的按钮</title>
06  <style>
07  * { margin:0; }                     /* 清除页面默认边距 */
08  #back_ground                        /* 设置背景 */
09  {
10      width:100%;                     /* 背景宽度与屏幕相同 */
11      min-height:660px;               /* 高度 */
12      background:-webkit-linear-gradient(top,#ccc,#000); /* 设置背景渐变颜色 */
13      float:left;
14  }
15  #top_pic                            /* 顶部背景图片 */
16  {
17      width:100%;                     /* 宽度 */
18      height:270px;
19      float:left;
```

```
20  }
21  .button                                  /* 设置各个按钮样式 */
22  {
23      width:84%;
24      height:50px;
25      margin:15px 8%;
26     border: 5px solid;
27     -webkit-border-radius: 15px;     /* 设置 webkit 浏览器下兼容的圆角 */
28     border-radius:15px;                /* 设置圆角 */
29      float:left;
30      font-size:32px;                   /* 设置字体 */
31      line-height:50px;
32      color:#FFF;
33      text-align:center;
34  }
35  #button_1                                /* 设置各个按钮的颜色 */
36  {
37      border-color:#FFF;                /* 按钮一边框为白色 */
38  }
39  #button_2
40  {
41      background-color:#F00;            /* 按钮二背景颜色为红色 */
42      background:#FF0;                  /* 按钮二背景颜色为黄色 */
43  }
44  #button_3
45  {
46      border-color:#345;                /* 按钮三边框颜色为灰色 */
47      background:-webkit-linear-gradient(top,#f00,#fff); /* 按钮三背景颜色渐变
*/
48  }
49  #button_4
50  {
51      border-color:#fff;                /* 按钮四边框颜色为白色 */
52      background:-webkit-linear-gradient(left,#f00,#fff);
                                         /* 按钮四背景颜色渐变，且方向与按钮三不同 */
53  }
54  </style>
55  </head>
56  <body>
```

```
57     <div id="back_ground">
58     <div id="top_pic">
59         <img src="pic.jpg" width="100%" height="100%" />
60       </div>
61     <div class="button" id="button_1">
62         按钮一
63       </div>
64       <div class="button" id="button_2">
65         按钮二
66       </div>
67       <div class="button" id="button_3">
68         按钮三
69       </div>
70       <div class="button" id="button_4">
71         按钮四
72       </div>
73     </div>
74 </body>
75 </html>
```

运行之后的界面如图 3-1 所示。



图 3-1　利用 HTML 5 实现渐变的背景和圆角的按钮

读者可以对比范例中的代码和图 3-1，现在对该范例的几个知识点分别进行讲解。

（1）在屏幕的最顶部，可以清楚地看到一张老虎的照片，一般在应用的顶部常常会采用类似的方式来展示一些图片或新闻。通常的做法是使用一个宽度与屏幕相同而高度固定的 div

标签在屏幕上占据一定的位置，然后使用 img 标签具体指定显示图片的内容。



虽然在 CSS 3 中已经有了背景图像的拉伸属性，但是由于在真实开发中顶部所使用的图片往往还需要加入一些滚动的特效，因此还是建议单独使用一个标签来显示图像。

（2）仔细观察图 3-1 的话能够看到页面的背景是有一定渐变的，这也是 CSS 3 中的一个新特性，可以通过设置 linear-gradient 的值来产生背景颜色渐变的效果，如范例第 12 行所示。



由于手机上大多使用 webkit 内核的浏览器，因此读者如果选择在 PC 上对本范例进行测试也许会与预期有些区别，这时就可以将范例中的-webkit-linear-gradient 修改为-moz-linear-gradient 和-ms-linear-gradient。

（3）除了渐变背景之外，圆角也是在 CSS 3 中最常被用到的一个属性，可以通过属性 border-radius 来设置圆角的半径，如范例中第 27 行和第 28 行所示。

（4）还有相当重要的一点就是当为移动设备设计界面时，由于各种屏幕尺寸差异较大，可能无法一一适配最完美的显示方案，因此在设计时要尽量让每个元素都单独占据一排的位置（如图 3-1 中的 4 个按钮）或者是两个、三个元素平分一排，这样能够让界面具有更强的“抗变形”能力。

3.2.2 利用 JavaScript 响应用户的操作

以上利用 HTML 5 和 CSS 3 实现了一个简单的界面，但是如果想要对它进行操作还要搭配上 JavaScript 才行。

【范例 3-2】利用 JavaScript 响应用户的操作。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>利用 JavaScript 响应用户的操作</title>
06  <style>
07  <!—省略部分 CSS 样式，具体内容可参考范例3-1  -->
08  </style>
09  <script>
10  function click_1() {
11      var a = document.getElementById("image");       // 获取 img 标签
12      a.src = "pic1.jpg";                              // 更换 img 中的图片
13  }
14  function click_2() {
15      alert("按钮二被按下");                                 // 弹出对话框
16  }
17  function click_3() {
```

```
18      alert("按钮三被按下");                                 // 弹出对话框
19  }
20  function click_4() {
21      alert("按钮四被按下");                                 // 弹出对话框
22  }
23  </script>
24  </head>
25  <body>
26      <div id="back_ground">
27          <div id="top_pic">
28              <img src="pic.jpg" width="100%" height="100%" id="image"/>
29          </div>
30          <div class="button" id="button_1" onClick="click_1();">
31              按钮一
32         </div>
33         <div class="button" id="button_2" onClick="click_2();">
34          按钮二
35         </div>
36         <div class="button" id="button_3" onClick="click_3();">
37          按钮三
38         </div>
39         <div class="button" id="button_4" onClick="click_4();">
40          按钮四
41         </div>
42     </div>
43  </body>
44  </html>
```

运行之后界面与图 3-1 相同，但是当用户点击“按钮一”时顶部的图像就会改变，而用户点击其他几个按钮时则会显示相应的对话框，如图 3-2 和图 3-3 所示。



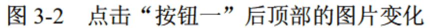


图 3-2　点击“按钮一”后顶部的图片变化



图 3-3　点击“按钮二”后弹出对话框

这是由于在范例的第 9~23 行声明了 4 个 JavaScript 函数，而在下面的第 30、33、36 和 39 行中使用 div 标签的 onClick 方法，对它们分别进行了调用。



在实际开发中对 JavaScript 只需要了解到如何使用它来弹出对话框或者显示内容到页面上就足够一般应用的开发了。当然随着层次的深入还会涉及执行效率等问题，不过这不是本书应当讨论的内容。

3.2.3 利用 CSS 3 生成动画

在前面的内容中已经通过 HTML 5 和 CSS 3 制作了一个简单的界面，并且使用 JavaScript 实现了对用户操作的响应。不过相对当前网上各种炫目的特效而言，这种例子似乎是有点令读者感觉到无聊。那好，在本节中将利用 HTML 5 来实现一些高级一点的功能。

“写轮眼”是漫画《火影忍者》中的一种技能，许多应用开发者会把它当作应用的开场动画来使用，本小节就以此为例使用 CSS 3 的动画功能实现“写轮眼”的效果。

【范例 3-3】利用 CSS 3 生成动画。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>利用 CSS 3生成动画</title>
06  <style>
07  body { background-color:#F00; }
08  #rec
09  {
10      width:300px;
11      height:700px;
12      margin:auto;
13  }
14  #title
15  {
16      width:300px;
17      height:40px;
18      font-size:24px;
19      color:#FFF;
20      text-align:center;
21      line-height:40px;
22      float:left;
23  }
24  #eye
```

```
25 {
26     margin-top:70px;
27     width:300px;
28     height:300px;
29     float:left;
30     background:url(bg.png);
31 }
32 #round
33 {
34     width:300px;
35     height:300px;
36     background:url(rote.png);
37     animation: myrole 5s;
38     -webkit-animation: myrole 5s;
39     animation-iteration-count:infinite;
40     -webkit-animation-iteration-count:infinite;
41 }
42 @keyframes myrole
43 {
44     from
45     {
46         transform: rotate(0deg);
47         -webkit-transform: rotate(0deg);
48     }
49     to
50     {
51         transform: rotate(360deg);
52         -webkit-transform: rotate(360deg);
53     }
54 }
55 @-webkit-keyframes myrole
56 {
57     from
58     {
59         transform: rotate(0deg);
60         -webkit-transform: rotate(0deg);
61     }
62     to
63     {
```

```
64          transform: rotate(360deg);
65          -webkit-transform: rotate(360deg);
66      }
67  }
68  </style>
69  </head>
70  <body>
71      <div id="rec">
72          <div id="title">CSS 3动画实现的写轮眼</div>
73          <div id="eye">
74                  <div id="round"></div>
75          </div>
76      </div>
77  </body>
78  </html>
```

运行之后的结果如图 3-4 所示。也许单纯使用图片无法展示出笔者真实看到的画面，但是当运行代码之后可以清楚地看到“写轮眼”确实在旋转。

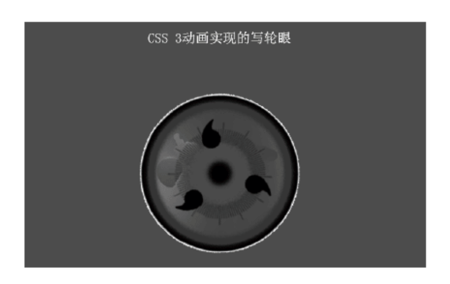


图 3-4 旋转的写轮眼

这种效果是依靠 CSS 的动画属性实现的，在范例的第 37 行和第 38 行是对屏幕上旋转部分所使用动画的定义，而这个动画的具体效果则是需要开发者根据需要进行设计的，如范例第 42~67 行所示。



在使用一些 CSS 3 中的属性时，有时会需要加入一些特定的前缀，比如-webkit-、-moz-等，而在介绍时为了方便一般都不会特别进行说明，比如范例第 37 行和第 38 行的 animation 属性和-webkit-属性其实是同样的作用，遇到类似的情况请读者根据范例进行简单理解即可。

通过范例中对动画进行定义部分的阅读可以发现，CSS 3 对动画的定义实际上就是在由 from 和 to 所包含的区间中，利用 CSS 的样式来声明元素在动画的开始和结束时所处的状态，然后由浏览器实现其间过程的变化。

还可以通过对元素的 animation-iteration-count 属性进行设置来控制动画播放的次数，由

于本范例中希望"写轮眼"能够一直旋转，因此就将它的值设置为 infinite。

3.2.4　利用 JavaScript 让"流氓兔"跑步

以上利用 CSS 3 实现了漫画《火影忍者》中"写轮眼"的效果，本小节依旧要实现一段动画，不过显示的内容就要可爱多了。

【范例 3-4】利用 JavaScript 让流氓兔跑步。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>利用 JavaScript 让流氓兔跑步</title>
06  <style>
07  body { background-color:#FFF; }          /* 背景颜色为白色 */
08  #rec
09  {
10      width:300px;
11      height:700px;
12      margin:auto;
13  }
14  #title                                   /* 动画顶部的标题 */
15  {
16      width:300px;
17      height:40px;
18      font-size:24px;
19      color:#FFF;
20      text-align:center;
21      line-height:40px;
22      float:left;
23  }
24  #tuzi                                    /* 流氓兔将在此元素中显示 */
25  {
26      width:300px;
27      height:300px;
28      float:left;
29  }
30  </style>
31  <script>
32  var step = 0;                            // 用于记录流氓兔运动的状态
```

```
33  setInterval(show,150);
34  function show() {
35      if(step%4 == 0) {
36          var a = document.getElementById("donghua");    // 获取流氓兔显示的元素
37          a.src= "IMG00.bmp";                           // 切换流氓兔显示的图片
38      }else if (step%4 == 1) {
39          var a = document.getElementById("donghua");
40          a.src= "IMG01.bmp";
41      }else if (step%4 == 2) {
42          var a = document.getElementById("donghua");
43          a.src= "IMG02.bmp";
44      }else if (step%4 == 3) {
45          var a = document.getElementById("donghua");
46          a.src= "IMG03.bmp";
47      }
48      step++;
49  }
50  </script>
51  </head>
52  <body>
53      <div id="rec">
54          <div id="title">利用 JavaScript 让流氓兔跑步</div>
55          <div id="tuzi">
56              <img src="IMG00.bmp" width="100%" height="100%" id="donghua" />
57          </div>
58      </div>
59  </body>
60  </html>
```

运行后的画面如图 3-5 所示，虽然无法利用图片来描述，但是可以保证的是，确实能够看到流氓兔在屏幕上跑起来了。



图 3-5　流氓兔跑步的画面

利用 JavaScript 显示动画时，会用到计数器函数 setInterval，它可以按照一定的周期重复

执行一系列的操作。比如本范例中就会以 150ms 为周期反复执行自定义函数 show()（如范例第 33 行所示）。

范例第 34~49 行是对 show 函数的定义，可以看出该函数利用了变量 step 除以 4 所得的余数来表示流氓兔跑步的动作，从而更换页面上应当显示的图片，给用户一种流氓兔在跑步的效果。

经证实，使用 JavaScript 实现的动画执行效率要远低于使用 CSS 3 的动画属性来显示动画，因此在开发游戏时要尽量使用 CSS 3 而不是 JavaScript，但是在本书中将会提到的一个“像素鸟”的例子仍然采用了 JavaScript 的方法，这是出于方便理解的目的，还请读者能够体谅。

3.3 关于界面设计

本章范例 3-1 中展示了一种使用 CSS 3 和 HTML 5 实现的界面，但是这个界面实际上有一点问题，那就是当屏幕过宽时可能会变形，如图 3-6 所示。



图 3-6　顶部的图片由于拉伸而严重变形

也许在图中由于使用了老虎而无法看出特别巨大的反差，那么假如顶部的图片换成了我们敬爱的“超人叔叔”，这时落差就非常明显了（如图 3-7 所示）。

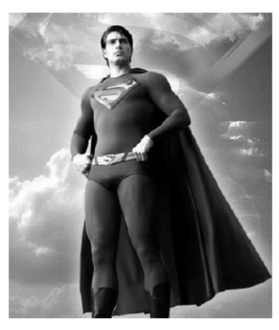
 

图 3-7　正常尺寸和被拉伸变形后的超人对比

对于这样的情况，目前确实找不到一种公认的“最好”方法来解决，但是却有几套备选方案：

- 利用 JavaScript 获取屏幕的尺寸并准备多套 CSS 样式文件，然后根据屏幕的尺寸来决定使用哪套 CSS 样式。
- 选择图片时尽量不使用有人物或者是电线杆等景物的图片，这样即使变形了也很难被发现。
- 直接利用 JavaScript 来保证图片的高度与宽度保持一定的比例。



作为一个懒人，笔者推荐使用第二种方案，但是实际上如果能将三种方案结合起来才是最好的方案。

除此之外还要考虑到顶部栏和尾部栏的样式，顶部栏和尾部栏都是应用中经常会用到的元素，许多开发者都担心，如果手机的像素密度越来越高，固定尺寸的顶部栏和尾部栏会不会由于像素密度的原因小到让用户看不清楚呢？

对于这个问题笔者也不好回答，因为这种情况在未来倒也不是不可能发生，但是就目前来看所谓最前沿的 4K 屏幕已经有些性能过剩了，因此几年内开发者们还没有必要为此而纠结。

将顶部栏和尾部栏的高度设置为 40px 是公认为比较合适的尺寸，为了保证显示的效果可以使用 position 将它们固定在屏幕的顶部和底部。

【范例 3-5】页面的顶部栏和尾部栏。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>页面的顶部栏和尾部栏</title>
06  <style>
07  * { margin:0px; }                    /* 清除页面自带边距 */
08  header                               /* 顶部栏 */
09  {
10      width:100%;
11      height:40px;                     /* 高度为40px */
12      background:#990;
13      position:fixed;                  /* 设置定位方式 */
14      top:0px;                         /* 使顶部栏位于屏幕顶端 */
15  }
16  footer                               /* 尾部栏 */
17  {
18      width:100%;
19      height:40px;                     /* 高度为40px */
20      background:#990;
21      position:fixed;                  /* 设置定位方式 */
22      bottom:0px;                      /* 使尾部栏位于屏幕底部 */
```

```
23  }
24  #title
25  {
26      width:100%;
27      height:40px;
28      line-height:40px;
29      color:#FFF;
30      text-align:center;
31      font-size:24px;
32  }
33  #content                              /* 页面中真正的内容 */
34  {
35      width:90%;
36      height:1600px;
37      margin:40px 5%;                   /* 设置外边距使被顶部栏和尾部栏遮挡的部分留白
*/
38  }
39  #content h1
40  {
41      margin:auto;
42      font-size:36px;
43      font-weight:bold;
44      text-align:center;
45      line-height:60px;
46  }
47  #line
48  {
49      width:100%;
50      height:1px;
51      background-color:#333;
52      float:left;
53  }
54  </style>
55  </head>
56  <body>
57      <header>
58          <div id="title">页面的顶部栏和尾部栏</div>
59      </header>
60      <div id="content">
61          <h1>此处显示内容</h1>
62          <div id="line"></div>
63      </div>
64      <footer>
65      </footer>
66  </body>
67  </html>
```

运行之后可以利用鼠标的滚轮配合 Ctrl 键来调整浏览器的放大和缩小，如图 3-8 和图 3-9 所示，可见在大多数情况下，40px 的高度已经足够用户看清屏幕了。



图 3-8　页面的顶部栏和尾部栏

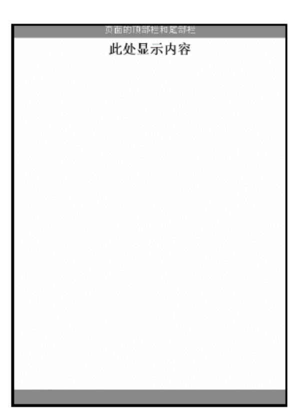


图 3-9　页面的顶部栏和尾部栏

通过范例中的 CSS 样式可以看出，顶部栏和尾部栏利用了将定位方式设置为 fixed 的方式，来保证最终这两个元素是相对于屏幕对齐的。在实际应用中还可以为这两个元素加入 z-index 属性，来保证它们不会被其他元素遮挡。

其次就是真正显示内容的 content 部分，由于它们的一部分内容是被顶部栏和尾部栏覆盖住的，因此如果在这些地方显示内容，结果就会有一部分内容无法显示，如图 3-10 所示，此时需要为它加入外边距，如范例第 37 行所示。

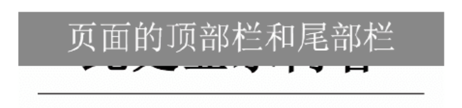


图 3-10　顶部栏的上边距被取消后一部分内容会被顶部栏遮盖住

至于其他的内容读者可以自己对比范例来学习，另外在实际使用时可能会有开发者喜欢为顶部栏加入一些阴影的特效，但是由于笔者艺术水平不高，尝试过后还是决定放弃了。

3.4 使用 jQuery Mobile 进行界面制作

上一节的内容介绍了在利用 HTML 5 实现应用界面时要注意的两个问题，其实相比界面变形，也许大多数开发者会更加烦恼另一个问题：如何设计界面。

与行业外到处鼓吹的“创新”不同，真正从事互联网行业的开发者应该会明白，大多数开发者其实更加信奉拿来主义，一些网上的开源框架很多可拿来作为自己应用的界面。在这些框架中 jQuery Mobile 是大多数开发者最热爱的一个。

下面将介绍使用 jQuery Mobile 进行界面制作的方法，首先可以去 jQuery Mobile 的官方网站（如图 3-11 所示，网址为：http://jquerymobile.com/）下载需要的 jQuery Mobile 压缩包。



图 3-11　jQuery Mobile 官方网站

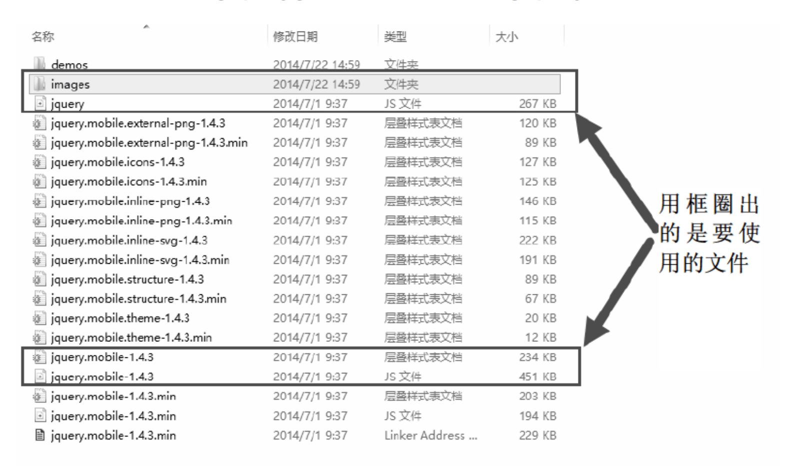
步骤 01　点击右侧的 Latest stable 按钮即可下载最新版本的 jQuery Mobile，当然也可以选择下载比较旧一点的版本。



提示

新版本的 jQuery Mobile 在 UI 上模仿了 iOS 7 的扁平化风格，并且加入了许多新的内容，但是由于国内所能找到的资料有限，并且一些新功能比较复杂，可以选择传统版本先来学习怎样使用 jQuery Mobile；但是在本节中由于只是为了体验，因此还是要选择最新版本。

步骤 02　解压下载好的文件，发现里面有好多文件，不过只需使用在图 3-12 中标出的几个文件就可以了（其中 jQuery.js 需要读者自己去 jQuery 官网下载）。



名称	修改日期	类型	大小
demos	2014/7/22 14:59	文件夹	
images	2014/7/22 14:59	文件夹	
jquery	2014/7/1 9:37	JS 文件	267 KB
jquery.mobile.external-png-1.4.3	2014/7/1 9:37	层叠样式表文档	120 KB
jquery.mobile.external-png-1.4.3.min	2014/7/1 9:37	层叠样式表文档	89 KB
jquery.mobile.icons-1.4.3	2014/7/1 9:37	层叠样式表文档	127 KB
jquery.mobile.icons-1.4.3.min	2014/7/1 9:37	层叠样式表文档	125 KB
jquery.mobile.inline-png-1.4.3	2014/7/1 9:37	层叠样式表文档	146 KB
jquery.mobile.inline-png-1.4.3.min	2014/7/1 9:37	层叠样式表文档	115 KB
jquery.mobile.inline-svg-1.4.3	2014/7/1 9:37	层叠样式表文档	222 KB
jquery.mobile.inline-svg-1.4.3.min	2014/7/1 9:37	层叠样式表文档	191 KB
jquery.mobile.structure-1.4.3	2014/7/1 9:37	层叠样式表文档	89 KB
jquery.mobile.structure-1.4.3.min	2014/7/1 9:37	层叠样式表文档	67 KB
jquery.mobile.theme-1.4.3	2014/7/1 9:37	层叠样式表文档	20 KB
jquery.mobile.theme-1.4.3.min	2014/7/1 9:37	层叠样式表文档	12 KB
jquery.mobile-1.4.3	2014/7/1 9:37	层叠样式表文档	234 KB
jquery.mobile-1.4.3	2014/7/1 9:37	JS 文件	451 KB
jquery.mobile-1.4.3.min	2014/7/1 9:37	层叠样式表文档	203 KB
jquery.mobile-1.4.3.min	2014/7/1 9:37	JS 文件	194 KB
jquery.mobile-1.4.3.min	2014/7/1 9:37	Linker Address ...	229 KB

图 3-12　使用 jQuery Mobile 时需要使用的几个文件

步骤 03　然后就可以使用 jQuery Mobile 进行开发了，如范例 3-6 所示。

【范例 3-6】使用 jQuery Mobile 实现的界面。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>使用 jQuery Mobile 实现的界面</title>
06  <link href="jquery.mobile-1.4.2.css" rel="stylesheet" />
07  <script src="jquery.js"></script>
08  <script src="jquery.mobile-1.4.2.js"></script>
09  </head>
10  <body>
11      <div data-role="page" data-theme="b">
12          <div data-role="header" data-position="fixed">
13              <h1>jQuery Mobile</h1>
14          </div>
15          <div data-role="content">
16              <a href="#" data-role="button">按钮一</a>
17              <a href="#" data-role="button">按钮二</a>
18              <a href="#" data-role="button">按钮三</a>
19              <a href="#" data-role="button">按钮四</a>
20              <a href="#" data-role="button">按钮五</a>
21              <a href="#" data-role="button">按钮六</a>
22          </div>
23          <div data-role="footer" data-position="fixed">
24          <div  data-role="navbar" data-position="fixed">
25                  <ul>
26                      <li><a href="#">微信</a></li>
27                      <li><a href="#">通讯录</a></li>
28                      <li><a href="#">找朋友</a></li>
29                      <li><a href="#">设置</a></li>
30                  </ul>
31              </div>
32          </div>
33      </div>
34  </body>
35  </html>
```

运行之后的界面如图 3-13 所示。



图 3-13　利用 jQuery Mobile 实现的界面

不知道各位读者有没有感受到 jQuery Mobile 的巨大优势，仅仅使用了 35 行代码就已经实现了要手写几百行代码才能够实现的样式，最重要的是凭借一般人的美术功底根本做不出这样精美的界面来。而且 jQuery Mobile 还有一些隐含的“福利”，比如说用户点击按钮时，按钮还会带有光晕的特效，并且在不同页面切换时也会带有默认的动画效果。



这里将不再对 jQuery Mobile 进行单独讲解了，因为要单独对 jQuery Mobile 进行介绍的话至少需要再写一整本厚厚的书才行，有兴趣的读者可以购买《构建跨平台 APP：jQuery Mobile 移动应用实战》进行学习。

3.5 编辑器的选择

本书首先推荐使用 Notepad++作为编辑器，因为在 Notepad++中能够通过 Ctrl 键和鼠标滚轮方便地切换字体的大小，这种做法会让开发者觉得非常舒服。但是有些时候 Notepad++却不是那么“合格”，比如说当一个页面中同时有 CSS、HTML 和 JavaScript 存在时，Notepad 并不能将它们全部识别并且高亮显示，而只能高亮标注其中的一种代码，这让笔者非常不适应（如图 3-14 所示）。

```
<head>
<script>
function open_news(i) {
    document.getElementById("news").style.display='none';
    document.getElementById("list").style.display='inline';
}
function back_list() {
    document.getElementById("list").style.display='none';
    document.getElementById("news").style.display='inline';
}
</script>
<style>
* { margin:0;}
#head
{
    width:100%;
    height:60px;
    background:#567;
    font-size:32px;
    color:#fff;
    line-height:60px;
    text-align:center;
```

图 3-14　页面中出现两种以上的脚本时 Notepad++的高亮支持

这里再推荐 Dreamweaver 作为各位读者的编辑器。也许读者正是通过它学会了制作网页，因此能够欣然接受这一建议，但是相信也有不少读者认为国内开发者利用 Dreamweaver 作为编辑器是一件非常“土鳖”的事情（论坛上经常为此事争论不休）。

类似的传闻笔者也听过不少，不过说起来不管是 Dreamweaver 还是 Cordova 甚至是 jQuery Mobile，它们都有共同的一个“老板”——Adobe。因此，本书推荐使用 Dreamweaver 就合情合理了。另外，在 Dreamweaver CS 6 中也提供了对 jQuery Mobile 和 Cordova 的支持（如图 3-15 所示）。



图 3-15　Dreamweaver CS6 为 Cordova 和 jQuery Mobile 提供的支持

3.6　Cordova 中的 API 能干什么

本章主要介绍进行 Cordova 开发前所需要做好的准备，那么现在是不是该介绍一些关于 Cordova 的事了呢？其实 Cordova 就是将 HTML 写成的页面显示出来，然后通过特定的

JavaScript 获取几组数据而已。

虽然说使用 Cordova 进行开发主要是依靠 HTML 各方面的知识，但对于一名 Cordova 开发者来说，最主要的还是 Cordova 各种 API 的用法。Cordova 为开发者提供了电池状态、相机、联系人、文件系统、音频等 API 接口，本节将一一介绍它们的功能和用途。

1. Accelerometer（加速度传感器）

Accelerometer 也就是人们所称的重力感应，可以用它来获取手机各个方向的加速度。比如，可以利用重力加速度约等于 10 的特点来获取当前手机的方向，可以在一些游戏中利用它和一些算法实现体感操作（如模拟用户对方向盘的操作）。

2. Camera（摄像头）

Camera 正如它的字面意思，可以通过它来获取摄像头采集到的信息，不过一般来说用处不大。

3. Capture（采集工具）

Capture 类似于录音机或录像机，可以用它录制音频、视频或者抓取图像上传到网络，也可以通过它获取来自网络的多媒体信息。Capture 多用在一些社交类应用中，如“人人网”的上传图片功能可以依靠它来简单实现。

4. Compass（指南针）

如果说加速度传感器是用来感应重力从而知道地面方向的话，那么指南针则可以获取东西南北的方向，可以通过它和加速度传感器、地理位置传感器配合实现一些很神奇的功能，如从用户当前正拍摄的照片中得知用户所在的方位。

这听上去非常难以实现，但是却并不是无法实现的，如从地理位置传感器上获取的信息表示用户正在海边；指南针又能够证明用户正面朝大海；那么甚至不需要对照片进行分析就可以判断出用户所拍摄照片的内容了。

5. Connection（网络连接）

Connection 判断用户所处的网络状态。

6. Contacts（联系人）

Contacts 对设备上的联系人进行增、删、改、查，是非常实用的一组 API。

7. Device（获取设备信息）

Device 可以获取设备的版本号、操作系统等信息。

8. Events（系统事件）

Events 是一些对系统时间进行响应的回调函数，比如在用户电量过低时发出通知，也可以对音量键或搜索键等功能进行响应。

9. File（文件管理系统）

可以通过 File 来管理手机上的文件，但是由于 Cordova 的执行效率问题，不建议读者尝试用它来开发一款文件管理器，甚至是简单的电子书阅读器。在应用中使用 File 来对文件进行一些简单的操作（比如在 txt 中保存一些留言或笔记）还是可以的。

10. Geolocation（地理位置传感器）

Geolocation 是通常所说的 GPS，社交软件中比较常用的一项功能，通常会配合其他传感器使用。

11. Media（媒体）

Media 用于对音频文件进行录制和播放，感觉不如采集工具实用，因此也比较鸡肋。

12. Notification（提醒）

Notification 可以调用设备的震动和蜂鸣器等功能，一般用来实现对用户操作的反馈，比如在游戏中当用户撞车后会有一连串的震动等。

13. Storage（本地存储）

Storage 是非常实用的一组 API，可以在本地使用简单的数据库功能，在实际开发中还可以缓存一些新闻或聊天记录等信息。

14. Globalization（全球化）

许多应用的用户是使用不同语言的人，如果应用受欢迎，不久就需要在多语言环境下使用。Globalization API 使全球化更方便，它允许应用查询操作系统的当前设置。开发者通过这个 API 判断用户使用的语言，然后使用适当的语言加载内容，还使用 API 中的方法更好地显示日期、时间、数字和货币单位。

15. InAppBrowser（内置浏览器）

InAppBrowser 的版本和 Cordova API 的版本更接近，允许在单独的窗口中加载网页。

16. Splashscreen（闪屏）

Cordova 提供了 Splashscreen API 能够用来在 Cordova 应用启动时显示自定义的闪屏。

3.7 小结

本章主要对 HTML 的一些应用方式进行了学习，并且预习了 Cordova 中各个 API 的作用，这样在今后的学习中读者才能够做到心中有数。当然，由于篇幅问题，本章所涉及的知识点都无法进行深入地讲解，读者会随着本书后面各章案例的增加来细化这些知识。下一章将开始学习 Cordova 中的各种 API。



第二篇

基础知识篇

第 4 章

◂ Cordova的本地事件 ▸

本章将引入手机应用中一个非常重要的概念——生命周期。可以说只有真正了解了 Cordova 的生命周期，才算是开始入门了。除了生命周期之外，本章还要教会读者一些在进行 Cordova 开发时的必备技能，比如输出信息的三种方法等。

本章的主要知识点有：

- 什么是软件的生命周期以及生命周期的存在意义。
- 什么是本地事件以及怎样用本地事件来使应用具有更强的交互性。
- Cordova 的生命周期与 Activity 的生命周期有什么区别和共性。
- Cordova 生命周期中 15 种本地事件的概念和应用。
- 在 Cordova 开发中获取调试信息的三种方法。

4.1 什么是生命周期

想要真正地理解 Cordova 应用开发的内涵，首先需要理解什么是生命周期。这在字面上其实非常容易理解，一个应用从开始运行→被手机加载→应用被退出之间的过程就称之为一个生命周期。为了使读者更容易理解，本节将以 Android 原生 SDK 中 Activity 类的生命周期结合 Eclipse 中的 LogCat 调试工具进行实战讲解。

4.1.1 Activity 的生命周期

先仔细地观察图 4-1 的内容，这是谷歌官方给出的 Activity 生命周期流程图，它包括了一个安卓应用从被创建到结束时所经历的各种事件。下面是 Activity 生命周期中所经历的各个过程。

（1）启动 Activity：系统将调用 onCreate 方法创建新的 Activity 对象，然后依次调用 onStart 方法和 onResume 方法使刚刚创建的 Activity 进入运行状态。

（2）暂停状态：当前的 Activity 被其他的 Activity 覆盖或手机锁屏，原 Activity 被放入后台，系统将调用 onPause 方法使 Activity 进入暂停状态。

（3）恢复状态：当处于暂停状态的 Activity 重新被运行时，系统将调用 onResume 方法

使之重新回到运行状态。

（4）后台状态：当用户点击 Home 键返回主屏，Activity 被保存在后台，系统将先调用 onPause 方法再调用 onStop 方法使 Activity 处于暂停状态。

（5）返回状态：当用户重新打开 Activity 时，系统会先调用 onRestar 方法再调用 onStar 方法，最后调用 onResume 方法使应用返回到运行状态。

（6）当前 Activity 处于被覆盖状态或者后台不可见状态，此时系统内存不足，进程中断，而后用户退回当前 Activity：再次调用 onCreate 方法、onStart 方法、onResume 方法，进入运行状态。

（7）用户退出当前 Activity：系统先调用 onPause 方法，然后调用 onStop 方法，最后调用 onDestory 方法，结束当前 Activity。

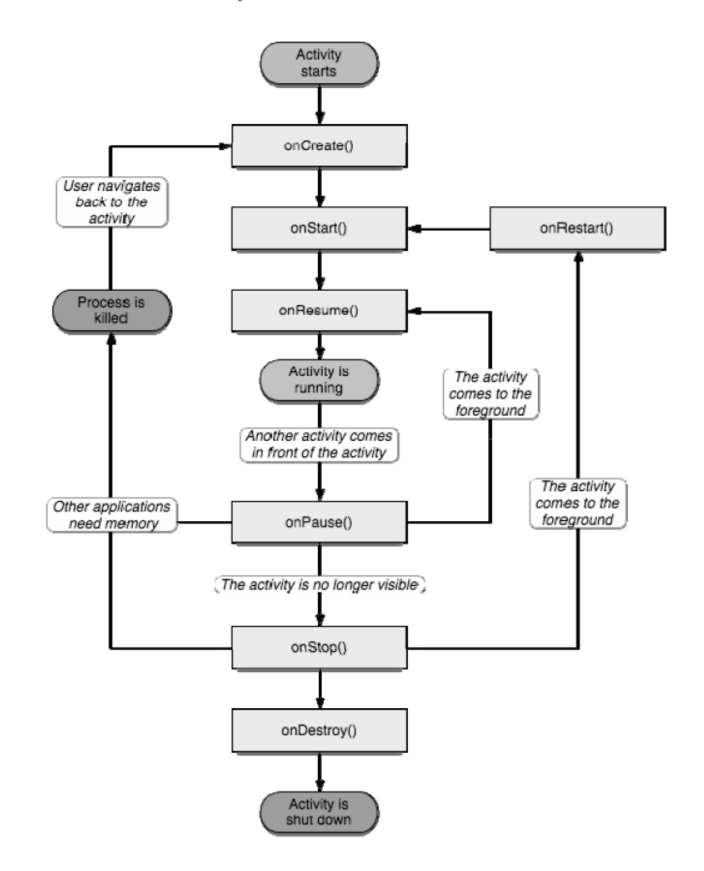


图 4-1　Activity 的生命周期

经过一番解释之后，相信读者已经能够看懂图 4-1 中的内容了，可是为什么要这样做呢？

众所周知，智能机相对于非智能手机的一个重要特点就在于，智能机具有“后台”，能同时运行多个程序。比如可以一边挂着 QQ，一边听音乐，同时浏览微博中的内容，而这时如果有人打电话进来，手机能够自动切换未接电话的界面，而这一切都是通过生命周期来实现的。

4.1.2　通过实例体验 Activity 的生命周期

以上介绍了 Activity 生命周期中的各个过程，本小节将以一个简单的实例来体验 Activity 生命周期中的各个事件。

在 Eclipse 中新建一个 Android 工程，命名为 example4_1，修改其 MainActivity 类中的内容如范例 4-1 所示。

【范例 4-1】Activity 生命周期的演示。

```
01 //此处省略若干个导入文件，由 Eclipse 自动生成
02 public class MainActivity extends Activity {   //类 MainActivity 继承类
Activity
03      String TAG="Activity 生命周期事件";
04      @Override
05      protected void onCreate(Bundle savedInstanceState) {
06          super.onCreate(savedInstanceState);
07          setContentView(R.layout.activity_main);
08          Log.e(TAG,"启动 onCreate 事件");
09      }
10      @Override
11      protected void onDestroy() {                        //重写 onDestroy 事件
12          // TODO Auto-generated method stub
13          super.onDestroy();
14          Log.e(TAG,"启动 onDestroy 事件");                  //在 LogCat 中显示记录
15      }
16      @Override
17      protected void onPause() {
18          // TODO Auto-generated method stub
19          super.onPause();
20          Log.e(TAG,"启动 onPause 事件");
21      }
22      @Override
23      protected void onRestart() {
24          // TODO Auto-generated method stub
25          super.onRestart();
26          Log.e(TAG,"启动 onRestart 事件");
27      }
28      @Override
29      protected void onResume() {
30          // TODO Auto-generated method stub
31          super.onResume();
```

```
32            Log.e(TAG,"启动 onResume 事件");
33        }
34        @Override
35        protected void onStart() {
36            // TODO Auto-generated method stub
37            super.onStart();
38            Log.e(TAG,"启动 onStart 事件");
39        }
40        @Override
41        protected void onStop() {
42            // TODO Auto-generated method stub
43            super.onStop();
44            Log.e(TAG,"启动 onStop 事件");
45        }
46    }
```

运行之后即可看到在 LogCat 窗口中显示出如图 4-2 所示的内容。



L...	Time	PID	TID	Tag	Text
E	01-19 02:51:07.658	1313	1313	Activity生命周期事件	启动onCreate事件
E	01-19 02:51:07.658	1313	1313	Activity生命周期事件	启动onStart事件
E	01-19 02:51:07.698	1313	1313	Activity生命周期事件	启动onResume事件
D	01-19 02:51:08.248	1313	1313	gralloc_goldfish	Emulator without GPU emulation detected.

图 4-2　Activity 启动时所经历的事件



可以通过设置过滤器来过滤 LogCat 中的信息，使之只显示与该 Activity 有关的记录，如图 4-3 所示，过滤后的内容如图 4-4 所示。

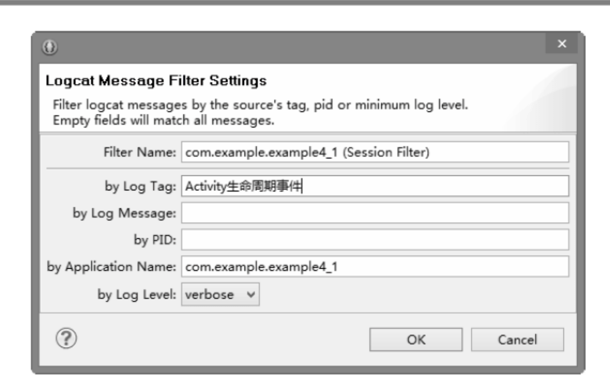


图 4-3　利用 Tag 标签过滤 LogCat 中的信息



图 4-4　过滤后的 LogCat 窗口

回过头来再看以上对启动 Activity 的描述，要启动一个 Activity 需要经历 onCreate、onStart、onResume 三个事件，在图 4-4 中可以确认这一点。可以通过对手机的进一步操作来验证以上的内容，比如点击 Home 键或返回键来观察 LogCat 中的日志记录。

在 Eclipse 的代码编辑区域点击鼠标右键，在弹出的菜单中，依次选择 Source|Override methods 命令可以看到一些其他的事件，如图 4-5 所示。

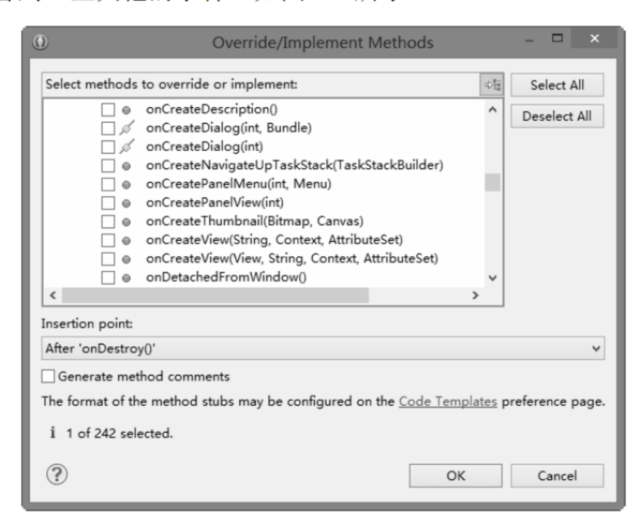


图 4-5　Activity 中的一些其他事件

这类事件大多对应着应用的某一特定操作或错误。比如事件 OnTitleChanged 就是在应用的标题被修改时由系统所发出的事件；再比如听音乐，音乐播放到 1 分 50 秒时突然打进来一个电话，系统就会对播放器的 Activity 使用 onPause 方法，同时启动一个接电话的 Activity，当用户接完电话后则又会通过 onRestar 方法和 onStar 方法返回音乐播放界面，并调整音乐播放进度为 1 分 50 秒。

4.1.3　Cordova 的生命周期

以上通过一个简单的例子演示了 Activity 的生命周期中事件的作用，即软件应用过程中发生了某操作后由系统产生的某种响应。与之类似的是，Cordova 中也有着生命周期和事件的概念。

在 Cordova 中，系统通过 JavaScript 截获来自于硬件的信息。目前，Cordova 可以处理包括网络、电量、音量、按钮等方面的信息。当使用 Cordova 编写的应用处于运行状态时，如果系统接收到了某种信号（比如用户按下了音量键），那么系统就能够做出相应的反馈。



Cordova 的生命周期只包括应用在屏幕中运行的一部分，当应用被暂停和重新运行时有 pause 事件和 resume 事件来与它们对应。

Cordova 的整个生命周期可以划分成 15 种不同的事件，如表 4-1 所示。

表 4-1 Cordova 生命周期中的事件

名称	说明
deviceready	当设备加载完毕后会触发该事件
pause	当程序被暂停到后台运行时会触发该事件
resume	当程序被从后台激活到前台运行时会触发该事件
online	当设备网络状态改变且是从网络断开状态切换到连接状态时触发此事件
offline	当设备网络状态改变且为从网络连接状态切换到断开状态时触发该事件
batterycritical	当设备电量过低超过了某个临界点时该事件被触发，临界点的值由设备决定，一般为 10%
batterylow	当设备剩余电量低于某个由开发者或用户指定的值时该事件被触发
batterystatus	当电池剩余电量发生 1%的改变时会触发该事件
backbutton	当用户点击“返回”按钮时该事件被触发
menubutton	当用户点击“菜单”按钮时会触发该事件
startcallbutton	当用户“按下”通话按钮时会触发该事件
endcallbutton	当用户点击“挂断”按钮时会触发该事件
volumedownbutton	当用户按下“音量减小”按钮时会触发该事件
volumeupbutton	当用户按下“音量增大”按钮时会触发该事件
searchbutton	当用户按下“搜索”按钮时触发该事件

看上去似乎有些混乱，既不是按照字母顺序也没有按照单词长度，但是按照表格右侧的解释来看却又有一定的规律。实际上这张表格是笔者经过了深思熟虑，按照事件的性质将它们重新分组后得来的结果，按照笔者的想法，这 15 个事件可以分为以下三类。

1. 程序加载事件

包括 deviceready、pause 和 resume 3 个事件，用于对程序的加载完毕（即生命周期的开始）、暂停和恢复进行处理。

2. 被动消息事件

用于对程序运行期间设备发生的一些变化进行处理，如电池电量的变化、网络断开等。包括以下 5 个事件：online、offline、batterycritical、batterylow、batterystatus。由于这些事件不是用户本人可以控制的，如电池的电量不可能由于用户的意愿而突然增加。因此笔者称其为被动消息事件。

3. 主动消息事件

包括 backbutton、menubutton、startcallbutton、endcallbutton、volumedownbutton、volumeupbutton 和 searchbutton 这 7 个事件，分别在用户按下相应的按钮时进行响应。

需要注意的是，并不是每部设备都具备这些按钮，比如在诺基亚最新发布的 Nokia X 手机中就只有正面的返回按钮以及侧面的音量键。因此如果想在这样的设备中对“搜索”按钮的操作进行处理是不可能的。而即使是在目前主流的安卓手机中一般也省略掉了“搜索键”而仅保留“返回”“菜单”以及在 Cordova 中没有提到的 Home 按钮，如图 4-6 中所示。

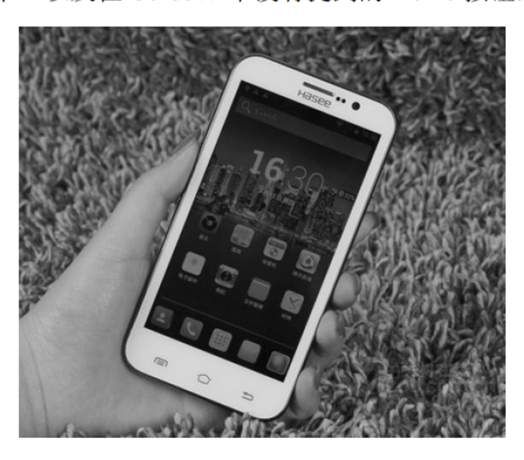


图 4-6　三按钮布局已成为当前安卓手机的主流设计

4.2 使用程序加载事件

在了解了 Cordova 中都有哪些事件之后，本节将开始对这些事件的用法进行详细的介绍。本节要介绍的是程序加载事件，也就是 deviceready、pause 和 resume 这 3 个事件。

【范例 4-2】程序加载事件的使用。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta charset="utf-8">
05  <title>程序加载事件的使用</title>
06  <!--引入 Cordova 脚本文件-->
07  <script src="cordova.js" type="text/javascript"/><script
type="text/javascript"/>
08      // 声明当设备加载完毕时的回调函数 onDevieReady
09      document.addEventListener("deviceready", onDeviceReady, false);
```

```
10        // 当设备加载完毕后就会执行该函数
11        function onDeviceReady() {
12            // 当该函数执行后，弹出对话框告诉用户设备已经加载完毕了
13            alert("设备加载完毕！");
14            // 一般来说需要保证在设备加载完毕之后再去执行其他操作
15            // 声明当程序被放置到后台暂停时执行的回调函数 onPause
16            document.addEventListener("pause", onPause, false);
17            // 声明当程序被从后台暂停状态恢复到前台执行时的回调函数 onResume
18            document.addEventListener("resume", onResume, false);
19        }
20        // 当程序被暂停时执行该函数
21        function onPause() {
22            // 当该函数被执行时，弹出对话框告诉用户该程序被暂停
23            alert("程序被暂停了！");
24        }
25        // 当程序被从暂停状态恢复时执行该函数
26        function onResume() {
27            // 当该函数被执行时弹出对话框告诉用户程序被恢复
28            alert("程序恢复运行");
29        }
30    </script>
31    </head>
32    <body>
33         <h1>程序加载事件的使用</h1>
34         <h3>程序开始运行后弹出对话框提示设备加载完毕</h3>
35         <h3>程序进入后台运行也弹出对话框提示程序被暂停</h3>
36         <h3>但当程序被恢复时却没有对话框弹出</h3>
37    </body>
38    </html>
```

程序运行之后，系统会自动对 Cordova 中的脚本进行加载，然后弹出如图 4-7 所示的界面，表明设备加载完毕。而当用户点击“返回”按钮或 Home 按钮时，也会弹出相应的对话框，如图 4-8 所示，但是不等笔者反应过来点击 OK 按钮，程序就已经被置入后台了。



图 4-7 设备加载完毕后弹出对话框

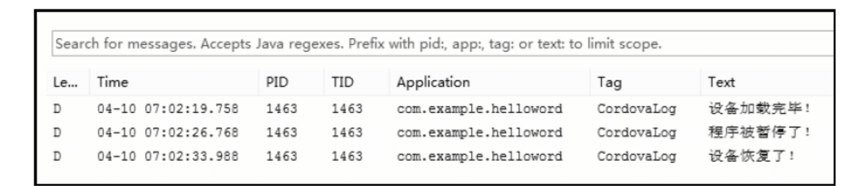


图 4-8 程序被暂停时同样弹出对话框提示

按照道理来说，如果此时再运行该程序也会弹出相应的对话框，但是真相是当再次运行该程序时却没有对话框弹出提示“程序被恢复”。这并不是写错了某段代码导致的，而是由于 Cordova 的某些特定调用关系所决定的，为了证明这一点，现在对范例的第 13、23 和 28 行做修改。在此次修改中放弃了利用 alert 方式弹出对话框，而是利用 console.log 方法使得当相应的函数被执行时，在 Eclipse 的 LogCat 面板上输出信息。

```
// 当设备加载完毕后就会执行该函数
function onDeviceReady() {
    console.log("设备加载完毕！");
}
// 当程序被暂停时执行该函数
function onPause() {
    console.log("程序被暂停了！");
}
// 当程序被从暂停状态恢复时执行该函数
function onResume() {
    console.log("设备恢复了！");
}
```

然后再运行该程序，在 LogCat 面板上输出的信息如图 4-9 所示。



Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope.

Le...	Time	PID	TID	Application	Tag	Text
D	04-10 07:02:19.758	1463	1463	com.example.helloword	CordovaLog	设备加载完毕！
D	04-10 07:02:26.768	1463	1463	com.example.helloword	CordovaLog	程序被暂停了！
D	04-10 07:02:33.988	1463	1463	com.example.helloword	CordovaLog	设备恢复了！

图 4-9　LogCat 中的信息

事实证明当程序从暂停状态下恢复时，如果调用一些 DOM 操作可能会得不到所预期的结果，这与安卓平台下 Cordova 的兼容性和运行效率等因素有关，在实际开发时需要特别注意这一点。

下面结合本范例来说明 Cordova 中各个事件的使用方法，通过范例的第 9、16 和 18 行可以看出，在 Cordova 中如果想对某个事件进行操作只需要按照“document.addEventListener ("eventname",function , false);”这样的格式进行定义就可以了。其中 eventname 是需要定义的事件名称，而 function 则是负责对该事件进行响应的自定义函数。



仔细观察范例可以发现一个有意思的问题，那就是对 pause 和 resume 两个事件的声明是在设备加载完毕之后进行的，这是一个非常好的习惯，每一个 Cordova 开发者都要努力适应这一点。

趁此机会再介绍一点额外的知识，那就是在 Cordova 中进行调试的方法。在程序开发时经常会遇到一些意外的错误，一般来说可以通过在特定的位置输出一些数据来验证程序出错的位置。这时就需要考虑使用什么方法来获取这些数据了。

对于习惯了 Web 开发的开发者来说，利用 alert 方法在对话框中弹出数据是一种非常方便的选择，但是在实际使用中这种方法并不是非常方便，因此就常常采用 console.log 方法输出信息以便于调试。

除了利用 console.log 方法之外，还有许多人喜欢利用 JavaScript 的 DOM 操作将结果直接输出在页面中，在某些情况下，比如需要统计某一变量在一段时间内的变化情况，这种方法也是非常实用的。

4.3 使用被动消息事件

本节将继续介绍 Cordova 中的另一类事件：被动消息事件。这类事件有一个共同的特点是非常难以调试。如果想要对 batterycritical 事件进行测试，可能就只有等到电量下降到某个程度才可以得到结果，尤其是这些数据还无法利用虚拟机进行模拟。另外，这类事件在使用上往往还比较鸡肋，比如开发者对电量事件的使用不外乎当电量低到一定程度时提醒用户充电，可是实际上安卓系统本身也会做同样的事情。因此开发者会尽量不使用这类事件。

虽然会避免使用它们，但是该介绍的知识点一个也不能少，这就让笔者突然想起了一句

古话叫做“敬鬼神而远之”。可能开发者在面对不少“不好用”的 API 时都是这样的态度吧。下面回到正题，请看范例 4-3 中的例子。

【范例 4-3】被动消息事件的使用。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta charset="utf-8">
05  <title>被动消息事件的使用</title>
06  <!--引入 Cordova 脚本文件-->
07  <script src="cordova.js" type="text/javascript"></script>
08  <script>
09      // 声明当设备加载完毕时的回调函数 onDevieReay
10      document.addEventListener("deviceready", onDeviceReady, false);
11      // 当设备加载完毕后就会执行该函数
12      function onDeviceReady() {
13          // 声明用于 online 事件的触发器函数
14          document.addEventListener("online", onOnline, false);
15          // 声明用于 offline 事件的触发器函数
16          document.addEventListener("offline", onOffline, false);
17          // 声明用于 batterycritical 事件的触发器函数
18          window.addEventListener("batterycritical", onBatterycritical,
false);
19          // 声明用于 batterylow 事件的触发器函数
20          window.addEventListener("batterylow", onBatterylow, false);
21          // 声明用于 batterystatus 事件的触发器函数
22          window.addEventListener("batterystatus", onBatterystatus, false);
23      }
24      // 当网络状态由断开切换到连接状态时触发此函数
25      function onOnline() {
26          alert("网络连接成功！");
27      }
28      // 当网络状态从连接切换到断开状态时触发此函数
29      function onOffline () {
30          alert("网络断开！");
31      }
32      // 处理电池电量不足的事件
33      function onBatterycritical (info) {
34          alert("电量过低，还剩：" + info.level + "%");
```

```
35        }
36        // 电池电量过低触发此函数
37        function onBatterylow (info) {
38            alert("电量过低,还剩:  " + info.level + "%");
39        }
40        // 电池状态发生改变触发此函数
41        function onBatterystatus (info) {
42            alert("电池状态改变了，剩余电量：" + info.level + "%");
43        }
44    </script>
45    </head>
46    <body>
47         <h1>被动消息事件的使用</h1>
48    </body>
49    </html>
```

运行之后的结果如图 4-10 所示。



图 4-10　电池状态改变时弹出对话框

有的读者也许会非常佩服笔者的敬业，可以为了找一张书中实用的图片而静等电量变化，这里要介绍一个独家窍门，那就是可以人为地创造 batterystatus 事件。比如拿着手机，启动范例程序，把手机充电器插上去，电量状态就改变了，再拔下来又改变一次。

这一组事件的使用方法与之前介绍的一样，但是在电池事件的回调函数中加入了参数，这里使用的 info 对象（如第 41 行所示）封装了相关电量的一些信息。它包含了两个属性，分别是 level（用于记录当前电量的百分值，取值范围为从 0~100）和 isPlugged（用于记录当前是否为充电状态）。

由于这类事件非常难以捕捉，在测试时可以将具体的函数操作写在 batterystatus 事件的回调函数中进行测试，测试完成后再移回到它该待的地方就可以了。

4.4 使用主动消息事件

与上一节介绍的 5 个事件相比，本节要介绍的 7 个事件简直是平易近人，因为直接通过点击按钮就可以对它们进行测试。它们的使用方法与之前介绍的 8 种事件完全一样，范例 4-4 是使用这些事件的一个例子。

【范例 4-4】使用主动消息事件。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta charset="utf-8">
05  <title>使用主动消息事件</title>
06  <!--引入 Cordova 脚本文件-->
07  <script src="cordova.js" type="text/javascript"></script>
08  <script>
09      // 声明当设备加载完毕时的回调函数 onDevieReay
10      document.addEventListener("deviceready", onDeviceReady, false);
11      // 当设备加载完毕后就会执行该函数
12      function onDeviceReady() {
13          // 声明返回按钮被按下时响应的回调函数
14          document.addEventListener("backbutton", onBackKeyDown, false);
15          // 声明菜单按钮被按下时响应的回调函数
16          document.addEventListener("menubutton", onMenuKeyDown, false);
17          // 声明通话按钮被按下时响应的回调函数
18          document.addEventListener("startcallbutton", onStartCallKeyDown,
false);
19          // 声明结束通话按钮被按下时响应的回调函数
20         document.addEventListener("endcallbutton", onEndCallKeyDown, false);
21          // 声明音量减按钮被按下时响应的回调函数
22          document.addEventListener("volumedownbutton", onVolumeDownKeyDown,
false);
23          // 声明音量增按钮被按下时响应的回调函数
24          document.addEventListener("volumeupbutton", onVolumeUpKeyDown,
false);
25          // 声明搜索按钮被按下时响应的回调函数
26          document.addEventListener("searchbutton", onSearchKeyDown, false);
27      }
28      // 当返回按钮被按下时触发该函数
29      function onBackKeyDown() {
30          alert("返回按钮被按下");
31      }
```

```
32        // 当菜单按钮被按下时触发该函数
33        function onMenuKeyDown() {
34            alert("菜单按钮被按下");
35        }
36        // 当通话按钮被按下时触发该函数
37        function onStartCallKeyDown() {
38            alert("通话按钮被按下");
39        }
40        // 当结束通话按钮被按下时触发该函数
41        function onEndCallKeyDown() {
42            alert("结束通话按钮被按下");
43        }
44        // 当音量减按钮被按下时触发该函数
45        function onVolumeDownKeyDown () {
46            alert("音量减按钮被按下");  //增按钮雷同，这里省略
47        }
48        // 当音量增按钮被按下时触发该函数
49        function onSearchKeyDown() {
50            alert("搜索按钮被按下");
51        }
52    </script>
53    </head>
54    <body>
55         <h1>使用主动消息事件</h1>
56    </body>
57    </html>
```

程序运行之后，依次按下各按钮可以看到相应的对话框弹出，最后的结果如图 4-11 和图 4-12 所示。



图 4-11　菜单按钮被按下



图 4-12　返回按钮被按下

其他按钮也是类似，不过需要注意的是，这里所说的按钮必须是实体或专门的虚拟按钮

才有效。比如图 4-13 所示的平板电脑的虚拟返回键是有效的，但是在一些应用中的“返回”按钮是不能触发 backbutton 事件的。



图 4-13　某款平板电脑中的虚拟键

最后需要说明的是，对音量增和音量减按钮的触发事件由于与系统本身的音量功能冲突，所以在大多数的场合下是无效的。另外，目前的大多数手机已经没有了通话按钮和通话结束按钮，因此这两个事件也无法进行测试了。



虽然 Cordova 为开发者提供了各种事件，但是除了 deviceready 在开发中有着广泛的应用之外，其他事件的应用价值不大。因此还请各位开发者能够以理性的态度使用它们。

4.5 小结

本章首先介绍了 Activity 的生命周期，通过案例让读者了解什么是生命周期。接着介绍了 Cordova 的生命周期有哪些事件，并将这些事件分了 3 类：程序加载事件、被动消息事件、主动消息事件。本章最后通过 3 个案例来介绍了如何使用这 3 类事件。

第 5 章

◂ 设备信息的获取 ▸

本章将介绍 Cordova 利用 API 获取设备信息的方法。利用这些 API，开发者可以获取与设备相关的一些信息，如手机的型号、Cordova 的版本，以及所使用的操作系统等信息，为开发者进一步实现其他的功能奠定基础。

本章的主要知识点有：

- Cordova 中 API 的使用方法。
- 如何将 API 获取的信息显示在屏幕上。
- 如何进一步使用 API。
- 如何美化利用 Cordova 开发的应用。

5.1 Cordova 获取设备信息

除了能够将 HTML 页面打包成可以直接安装运行的 APP 外，Cordova 的一个最大优势在于可以通过 JavaScript 调用设备来访问设备上的硬件信息，从而实现一些原本只有依靠原生 SDK 才能够达到的目的。范例 5-1 就展示了一个利用 API 来获取设备信息的例子。

【范例 5-1】利用 Cordova 获取设备信息。

先在命令行进入到项目文件夹下，执行以下命令添加插件：

```
cordova plugin add cordova-plugin-device
```

```
01  <!DOCTYPE html>
02  <html>
03  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
04  <head>
05  <script src="cordova.js" type="text/javascript" charset="utf-8"></script>
06  <script type="text/javascript" charset="utf-8">
07      //设置触发器函数 onDeviceReady()
08      document.addEventListener("deviceready", onDeviceReady, false);
09      // Cordova 加载完毕，现在可以安全地调用 Covdova 方法
```

```
10      function onDeviceReady() {
11         // 现在可以安全使用 Cordova API
12         //获取页面中 id 为 deviceProperties 的元素
13           var element = document.getElementById('deviceProperties');
14           //将获取的设备信息写入到页面元素中
15           element.innerHTML = '设备名称: '     + device.name     + '<br />' +
16                                 'Cordova 版本: ' + device.cordova + '<br />' +
17                                 '操作系统: ' + device.platform + '<br />' +
18                                 '设备编号: '     + device.uuid     + '<br />' +
19                                 '操作系统版本: '  + device.version  + '<br />';
20      }
21
22  </script>
23  </head>
24  <body bgcolor="#E4E4E4">
25    <h1>cordova</h1>
26      <h4 id="deviceProperties"></h4>
27  </body>
28  </html>
```

使用 Cordova 编译之后，运行结果如图 5-1 所示。



图 5-1　利用 Cordova 显示设备信息

通过本例可以看出，使用 Cordova 中 API 的方法与使用原生 JavaScript 和 HTML 的方法没有什么不同，只是多了一些事先定义好的类，如范例第 15 行中的 device。

Cordova 定义了一个类 device，其中包含了 name、cordova、platform、uuid、version 这 5

个成员变量，分别存放设备的设备名称、Cordova 版本号、操作系统、设备编号和操作系统版本，使用时可以直接引用。



不同的 Cordova 版本，此处测试会略有不同。

在范例的第 15~19 行就是使用了 innerHTML 操作来将 device 类中的信息显示到页面元素中，而页面元素是在第 13 行中利用元素的 id 属性来获取的。熟悉 JavaScript 的读者可以发现，这与 JavaScript 的 DOM 操作完全相同。

5.2 device 类的异常情况

上一节介绍了一个利用 Cordova 获取设备信息的例子，但是细心的读者一定会发现，在设备名称处显示的内容是 undefined，这又是为什么呢？要解决这个疑问就必须深入了解 device 类中每个属性的值。

name 属性用来描述设备的名称，这个值由制造商决定，也就是说，即使是同一款手机，由于生产批次不同，最终的设备名称也可能不同。但是还会有一些例外的情况，比如在 Android 设备中常常获取到的是设备的产品名称，大名鼎鼎的 Nexus 系列获取的名称就是 Passion，而在苹果的设备中该属性获取的值是 iTunes 中设定的用户名。上一节的范例由于采用的是虚拟机，所以无法获取设备名称，因而显示成了 undefined。

Cordova 属性的值比较固定，一般都是当前所使用的 Cordova 的版本号，范例之所以读取出错也是由于虚拟机的缘故。



其实该属性并没有什么使用价值，因此无论正确还是错误，影响都不大。

platform 属性用来保存设备所使用的操作系统的名称，在范例中非常正确地识别出了设备的操作系统为 iOS。当然如果设备采用的安卓的设备就会显示 Android。另外黑莓手机中显示的该属性是所使用的 rom 版本号，这对没有使用过黑莓的开发者来说也许会比较陌生，但是请放心，黑莓的用户都能够理解这组编号。

uuid 的值对于用户来说是非常重要的，因为这个值对于每台设备来说就像是身份证一样，它是唯一的，在开发一些手机防盗应用时会经常使用。



不只是移动设备，电脑的 CPU、网卡、显卡等设备中也会用到该值，图 5-2 是在电脑的 BIOS 中查看电脑的 uuid 值。

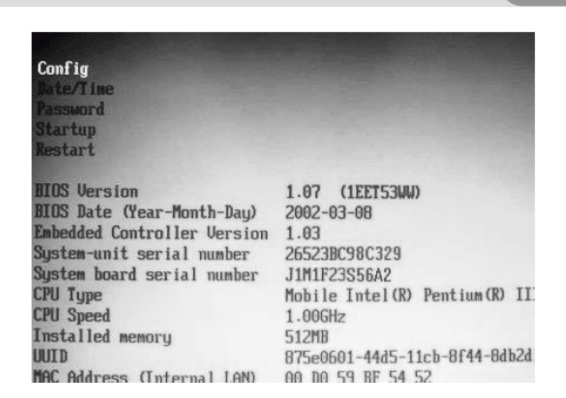


图 5-2　在 BIOS 中查看电脑的 uuid

version 属性用来显示设备的具体版本号，比如范例中的 iOS 系统为 10.0.1 版本，知道了该属性可以根据设备的版本号来对一些功能加以取舍，来保证 APP 运行的流畅性。

5.3 实战：用 Cordova 制作一个简单的应用

前面几节介绍了使用 Cordova 获取设备信息的方法，可是仅仅通过一个简单的例子显然是无法直接应用到真实开发中的。因此本节就来做一个复习，利用已经学过的知识来实现一个查看设备信息的简单应用。

本次的任务非常明确，就是要实现一个简单的页面，在屏幕偏下方的位置有一个按键，当用户按下它时将会在屏幕的上半部分显示出设备的信息，同时要保证应用的流畅性和美观。

5.3.1 界面设计及实现

由于这款应用功能非常单一，而且只有一个页面，因此页面中的样式利用自定义的 CSS 来实现，既不会太麻烦也能够保证应用运行的高效性。为了保证良好的交互性，应用要对用户的操作做出简单的反馈，如点击按钮时按钮会变色、显示设备信息前会有一段动画作为缓冲等。

范例 5-2 是该项目使用的基本界面。

【范例 5-2】应用界面的基本实现。

```
01  <!DOCTYPE html>
02  <html>
03  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
04  <head>
05  <script src="cordova.js" type="text/javascript" charset="utf-8"></script>
06  <script type="text/javascript" charset="utf-8">
```

```
07  </script>
08  <style>
09  /*CSS 中利用*表示所有样式，将页面内的全部边距清零*/
10  *
11  {
12      margin:0;
13      padding:0;
14  }
15  /*使用 CSS3的渐变效果设置背景颜色*/
16  body
17  {
18      height:600px;
19     background:-webkit-linear-gradient(yellow,orange,red,green,blue,purple);
20  }
21  /*重写 h2标签的字体颜色*/
22  h2{ color:#fff; margin:5px; }
23  /*利用绝对定位为按钮提供可靠的位置*/
24  .button_rec
25  {
26      width:100%;
27      height:80px;
28      bottom:50px;
29      position:absolute;
30  }
31  /*设置按钮的大小和位置*/
32  .button_main
33  {
34      width:100%;
35      height:80px;
36      position:relative;
37  }
38  /*按钮的样式*/
39  .button
40  {
41      width:70%;
42      height:80px; margin-left:15%;
43      background:-webkit-radial-gradient(white,yellow);
44      border:9px solid #F00;
45      border-radius:25px;
```

```
46      -webkit-border-radius:25px;
47      font-weight:900;
48      font-size:32px;
49      line-height:80px;
50      text-align:center;
51  }
52  /*使按钮在被点击时改变为红色，使应用具有更强的交互性*/
53  .button:active
54  {
55      background:#f00;
56  }
57  </style>
58  </head>
59  <body>
60      <h2>点击屏幕下方的按钮，系统将会自动获取设备信息。</h2>
61      <div class="button_rec">
62          <div class="button_main">
63              <div class="button">
64                  获取设备信息
65              </div>
66          </div>
67      <div>
68  </body>
69  </html>
```

编译之后，运行结果如图 5-3 所示。虽然看上去比较有山寨感，也许很难与美观这个词联系到一起，但是它已经具备了一个完整的应用界面所需要的各种要素。

上述代码最先实现的是背景，由于本例非常简单，因此可以直接为 body 加入 background 属性。不过在这之前先要做一点小小的准备来清除浏览器默认的一些设置，如范例第 10~14 行所示。这是由于浏览器默认会带有一些样式（如元素的外边距），这就导致了页面的四周可能会有一段的空白，因此要坚决取消。

在 body 标签中有一个 height 属性要特别注意，由于在 WebKit 内核的浏览器中使用渐变背景色与理论百分比设置的长款无法完美地兼容，因此就只能设置一个大概的高度。虽然说设备的屏幕高度是无法预期的，但是可以设置一个比屏幕高度要大一些的高度，这样就能够保证不影响屏幕的效果（在本例中紫色的部分就由于超出了屏幕范围而没有显示，并不影响整体效果）。



实际开发中也可以根据经常出现的分辨率来分别编写几组 CSS，然后通过获取设备屏幕的尺寸来选择使用哪一组。



图 5-3　实现后的应用界面

范例第 39~51 行是按钮的 CSS 样式，它首先使用了 CSS 3 中的圆角属性，然后又为圆角加入了一个边框，保证按钮的颜色能够与周围的背景色有一个明显的区分。

为了增强用户点击按钮的交互性，在范例的第 53 行又为按钮加入了一个伪类，使按钮在被用户点击后改变颜色，这样用户能够清楚地感觉到应用做出了回应。图 5-4 为按钮被点击时的样子。



在实际开发时还可以为按钮加入一些图片或阴影效果。



图 5-4　当按钮被点击时变为红色

5.3.2　为应用加入功能

以上已经实现了利用 HTML 5 与 CSS 3 来完成应用的界面，本小节将为该界面加入获取设备信息的功能。许多读者一定会认为本小节是在画蛇添足，因为他们会认为只要把范例 5-2 中的代码复制到界面中就可以了，实际上却并不是这样，这是为什么呢？请看范例 5-3 中的代码。

【范例 5-3】为应用加入功能。

```
01  <!DOCTYPE html>
02  <html>
03  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
04  <head>
05  <script src="cordova.js" type="text/javascript" charset="utf-8"></script>
06  <script type="text/javascript" charset="utf-8">
07  </script>
08  <script type="text/javascript" charset="utf-8">
09      //定义一个标志位，若值为0表示设备未完成加载
10      var isready=0;
11      //设置触发器函数 onDeviceReady()
12      document.addEventListener("deviceready", onDeviceReady, false);
13      // Cordova 加载完毕，现在可以安全地调用 Cordova 方法
14      function onDeviceReady() {
15         // 现在可以安全使用 Cordova API
16         //标志位表明设备已经准备好了
17         isready=1;
18      }
19      function get_info()
20      {
21          if(isready==1)
22          {
23              //获取页面中 id 为 deviceProperties 的元素
24              var element = document.getElementById('deviceProperties');
25              //将获取的设备信息写入到页面元素中
26              element.innerHTML = '设备名称:'   + device.name    + '<br />' +
27                                  'Cordova 版本:' + device.cordova + '<br />' +
28                                  '操作系统:' + device.platform + '<br />' +
29                                  '设备编号:'   + device.uuid     + '<br />' +
30                                  '操作系统版本:' + device.version  + '<br />';
31          }
32      }
33  </script>
34  <style>
35  /*CSS 中利用*表示所有样式，将页面内的全部边距清零*/
36  *
37  {
38      margin:0;
```

```
39        padding:0;
40    }
41    /*使用 CSS3的渐变效果设置背景颜色*/
42    body
43    {
44        height:600px;
45       background:-webkit-linear-gradient(yellow,orange,red,green,blue,purple);
46    }
47    /*重写 h2标签的字体颜色*/
48    h2{ color:#fff; margin:5px; }
49    /*利用绝对定位为按钮提供可靠的位置*/
50    .button_rec
51    {
52        width:100%;
53        height:80px;
54        bottom:50px;
55        position:absolute;
56    }
57    /*设置按钮的大小和位置*/
58    .button_main
59    {
60        width:100%;
61        height:80px;
62        position:relative;
63    }
64    /*按钮的样式*/
65    .button
66    {
67        width:70%;
68        height:80px; margin-left:15%;
69        background:-webkit-radial-gradient(white,yellow);
70        border:9px solid #F00;
71        border-radius:25px;
72        -webkit-border-radius:25px;
73        font-weight:900;
74        font-size:32px;
75        line-height:80px;
76        text-align:center;
77    }
78    /*使按钮在被点击时改变为红色，使应用具有更强的交互性*/
79    .button:active
80    {
81        background:#f00;
82    }
83    </style>
84    </head>
85    <body>
```

```
86      <h2 id="deviceProperties">点击屏幕下方的按钮，系统将会自动获取设备信息。</h2>
87      <div class="button_rec">
88          <div class="button_main">
89              <div class="button" onclick="get_info();">
90                  获取设备信息
91              </div>
92          </div>
93      <div>
94  </body>
95  </html>
```

编译并运行之后，按下按钮获取设备信息后的结果如图 5-5 所示。



图 5-5　应用运行后的效果

观察范例的第 89 行，此处为按钮加入了一个 onclick 事件，当它被点击时会运行第 19 行处的 get_info()函数。由于这样并不能知道系统是否已经加载好了设备，因此必须在页面刚刚被载入时加入一个标志变量（如第 10 行）。

在代码第 14 行，由于设备已经完成了加载，因此可以在函数中为标志变量赋值，如第 17 行所示。这样，如果设备没有完成加载，点击按钮就是无效的了。

在实际开发中，也许还要考虑到用户在设备加载完成前就点击了按钮长时间等待后不耐烦的情况，这时就可以再加入一个新的函数，利用一个计时器函数不断地读取标志变量的值，同时在屏幕上方输出“请用户等待”的信息。当读取到 isready 变量值为 1 时，再执行将设备信息显示出来的操作。



在实际开发尤其是商业级别的开发中，常常需要考虑到一些非常极端的情况。

至此，一个简单的设备信息查看 APP 就算是完成了，读者如果有兴趣也可以在接下来的学习中继续完善，为它加入更多的特效或功能。

5.4 小结

本章主要学习了利用 Cordova 来获取设备信息的方法，读者从中可以看出 Cordova 中的操作几乎是完全按照原生 JavaScript 的语法来进行的，这是它的一个巨大的优势。除此之外，本章还进行了一次实际的开发实战，通过这次实战能够学到如何将知识点运用到实际 APP 中。本章的内容虽然简单，但是很关键，希望读者都能动手亲自实践这些范例。

第 6 章

◀ 通讯录信息的获取 ▶

通讯录不仅是手机用户最常接触的一个功能，也是手机最基础的功能之一，市面上也不乏类似的应用。即使是这样，开发者在通讯录上总能够找到新的亮点来开发出有创意、有前景的应用，所以掌握 Cordova 对通讯录信息的获取非常重要。本章将制作一个简单的通讯录管理软件从而使读者能够将知识应用到实际开发中去。

本章的主要知识点有：

- 通讯录信息的获取。
- 如何利用 JavaScript 获取通讯录中的联系人信息。
- 利用 Cordova 在通讯录中加入一个新的联系人。
- 如何从通讯录中删除一个联系人。
- 如何使用 Cordova 在通讯录中查找特定的联系人。

6.1 创建一个 Contact 对象

本节学习使用 Cordova 操作手机通讯录。Cordova 把有关通讯录的信息封装在一个 Contact 类中，因此在对通讯录进行操作前首先要创建一个 Contact 对象。

首先在命令行执行命令添加插件：cordova plugin add cordova-plugin-contacts

【范例 6-1】创建一个 Contact 对象。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <script src="cordova.js" type="text/javascript" charset="utf-8"></script>
06  <script type="text/javascript" charset="utf-8">
07      // 等待加载 Cordova
08      document.addEventListener("deviceready", onDeviceReady, false);
09      // Cordova 加载完毕
10      function onDeviceReady() {
```

```
11            //此处可以安全使用 Cordova 中的 API
12            //创建一个新的 Contact 对象
13            var myContact = navigator.contacts.create({"displayName": "Test
User"});
14            //设置 Contact 的 gender 属性
15            myContact.gender = "male";
16            //用对话框显示获得的 Contact 对象
17            alert("The contact, " + myContact.displayName + ", is of the " +
myContact.gender + " gender");
18            myContact.save();
19        }
20    </script>
21    </head>
22    <body>
23        <h1>Cordova 中的联系人</h1>
24        <p>创建一个 Contact 对象</p>
25    </body>
26    </html>
```

编译运行之后如图 6-1 所示，对话框中显示了刚刚创建的 Contact 对象的信息。

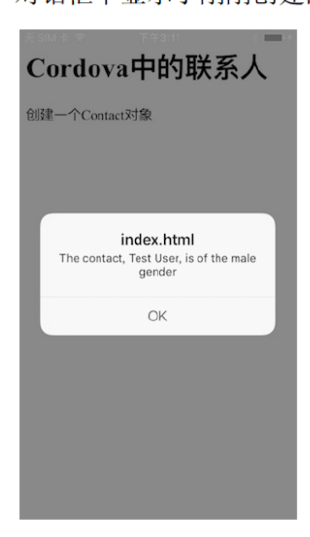


图 6-1　显示创建出的 Contact 对象

在 Cordova 中使用类 Contact 来管理通讯录，每次要对通讯录进行操作前，首先要使用语句 var contact = navigator.contacts.create(properties)来实例化一个 Contact 对象，如范例中第 13 行所示。



Cordova 官方给出的说明中使用了 var contact = navigator.service.contacts.create(properties)，但是该语句仅在 1.0 版本的 Cordova 中有效，在之后的版本中去掉了 service，因此如果照抄官方 demo，在运行时很可能出错。

范例第 15 行还设置了 Contact 对象的 gender 属性为 male，在本例中并没有实际的意义，只是为了展示对 Contact 对象进行操作的方法。结合图 6-1 中的内容可以清楚地看到经过这番操作之后 Contact 对象的属性。

6.2 利用 find()方法查询通讯录

上一节通过 Contact 中的 create()方法创建了一个新的 Contact 对象，本节将介绍 Contact 对象中的另一个方法 find()。它用来查询通讯录中的数据，能够返回一个或多个包含指定字段的 Contact 对象（即一个类型为 Contact 的数组）。范例 6-2 就是一个使用 find()方法查询通讯录的例子。

【范例 6-2】find()方法的使用。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <script src="cordova.js" type="text/javascript" charset="utf-8"></script>
06  <script type="text/javascript" charset="utf-8">
07      var isReady=0;
08      // 等待加载 Cordova
09      document.addEventListener("deviceready", onDeviceReady, false);
10      // Cordova 加载完毕
11      function onDeviceReady() {
12          //此处可以安全使用 Cordova 中的 API，将标志变量置为1
13          isReady=1;
14      }
15      //获取联系人结果集
16      function onSuccess(contacts) {
17          for (var i=0; i<contacts.length; i++) {
18              alert("联系人是： " + contacts[i].displayName);
19          }
20      }
21
```

```
22      // onError: 获得联系人失败
23      function onError() {
24          alert('onError!');
25      }
26      function find(){
27          if(isReady){
28              //查询资料中含有 Cat 的联系人
29              var options = new ContactFindOptions();
30              options.filter="Cat";
31              var fields = ["displayName", "name"];
32              navigator.contacts.find(fields, onSuccess, onError, options);
33          }
34      }
35  </script>
36  </head>
37  <body>
38     <h1 onclick="find()">Cordova 中的联系人</h1>
39  </body>
40  </html>
```

编译运行之后，点击屏幕上的“Cordova 中的联系人”几个字得到如图 6-2 所示的界面。



在运行之前一定要保证通讯录中有联系人，不然无法得到相应的结果。

范例第 32 行使用 find()方法来查找资料中包含 Cat 的联系人，在使用该方法之前必须要创建两个参数 options（第 29 行）和 fields（第 31 行）。options 参数中包含了要查询的内容，比如本例中 options.filter 的值为 Cat。fields 参数则是定义此次查找的目标字段，即 displayName 和 name。



按照西方的命名习惯，联系人的姓和名是分开的，而 displayName 字段就可以看作是姓和名的组合体。

除了这两个参数，使用 find()方法时还要在其中另外定义两个参数，其中 onSuccess 用来执行查询操作。在执行 find()方法之后，符合条件的 Contact 信息将会存放在一个数组中，可以在 onSuccess 函数中对它们进行处理。onError 函数则是用来指定 find()方法出错时的提示。



图 6-2　查询含有 Cat 的联系人的结果

此外还要知道，第 32 行查找联系人的 API 使用了多个参数，其中分别使用了 onSuccess 和 onError 函数来对 API 调用成功和失败的情况进行处理，而不是直接定义 API 的返回值。Cordova 提供的 API 中大多使用了这样的方法。

6.3 Contact 对象的属性

在之前的内容中曾经多次用到 Contact 对象，也学习了 Contact 的两个方法 create()和 find()，但是作为一个类，Contact 又有哪些属性呢？这是本节将要介绍的主要内容。

打开手机的通讯录，新建一个联系人会打开如图 6-3 所示的界面。可以看到每一个联系人都包括了姓名、电话、住址、邮箱等内容，而这些内容就是存储在类 Contact 对象中的信息。表 6-1 中记录了 Contact 各个属性的名称和意义。

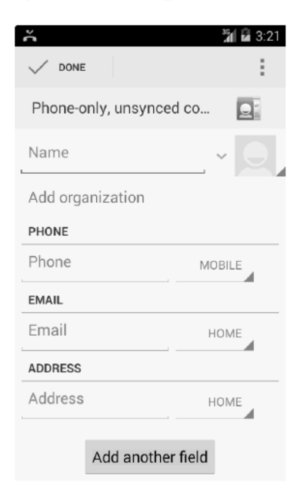


图 6-3　在安卓手机中新建一个联系人

表 6-1　Contact 中的属性

属性名称	类型	介绍
id	String	联系人的唯一标识符
displayname	String	联系人最终显示的名称
name	String	联系人的姓名
nickname	String	联系人的昵称
phonenumbers	String 型数组	联系人的电话
emails	Contactfield 型数组	联系人的电子邮箱
addresses	ContctAddrsses 型数组	联系人的全部住址
ims	Contactfield 型数组	联系人的全部 IM 地址
organizations	ContactOrganization 型数组	联系人的所在单位、组织等信息
birthday	date	联系人的生日
note	String	联系人的备忘、注释等信息
photos	Contactfield 型数组	联系人的全部头像信息
categories	Contactfield 型数组	用户对联系人添加的自定义信息
urls	Contactfield 型数组	联系人的主页等信息



通过这些属性就可以简单地对联系人的信息进行查看、修改、删除等操作。

范例 6-3 将利用本节介绍过的属性新建一个联系人信息并保存。

【范例 6-3】利用 Contact 对象新建一个联系人。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <script src="cordova.js" type="text/javascript" charset="utf-8"></script>
06  <script type="text/javascript" charset="utf-8">
07      var isReady=0;
08      // 等待加载 Cordova
09      document.addEventListener("deviceready", onDeviceReady, false);
10
11      // Cordova 加载完毕
12      function onDeviceReady() {
13          //此处可以安全使用 Cordova 中的 API，将标志变量置为1
```

```
14          isReady=1;
15      }
16      function create(){
17          if(isReady){
18              //新建一个 Contact 对象
19              var contact = navigator.contacts.create();
20              contact.displayName = "Jack";
21              contact.nickname = "Jack";     //同时指定以支持所有设备
22              // 填充一些字段
23              var name = new ContactName();
24              name.givenName = "Jane";
25              name.familyName = "Doe";
26              contact.name = name;
27              //保存
28              contact.save(onSuccess,onError);
29          }
30      }
31      function onSuccess(){
32          alert("新建联系人成功");
33      }
34      function onError(){
35          alert("新建联系人失败");
36      }
37  </script>
38  </head>
39  <body>
40      <h1 onclick="create()">新建一个联系人</h1>
41  </body>
42  </html>
```

编译运行之后，点击屏幕上的“新建一个联系人”，可以弹出一个对话框，内容为“新建联系人成功”，如图 6-4 所示。之后查看手机的通讯录可以看到确实在通讯录中增加了一个新的联系人 Jane Doe，如图 6-5 所示。



图 6-4　提示新建联系人成功

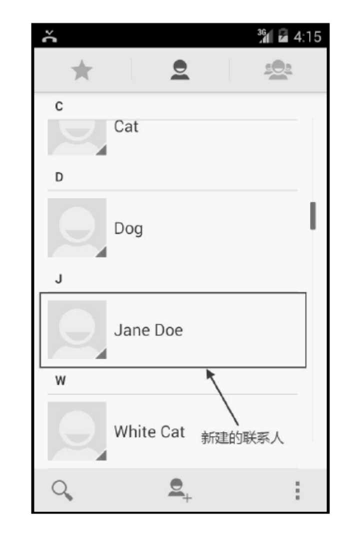


图 6-5　新建立的联系人为 Jane Doe

本范例要特别注意的是在第 23~26 行对 name 属性的保存，必须新建一个 ContactName 对象。此外建议在创建 displayname 和 nickname 属性时尽量使它们一致，因为在某些系统中这两个属性是混用的，只有这样才能保证应用的兼容性。第 28 行使用了 save()方法来保存刚刚创建的 Contact 对象，使它被记录到手机中。



读者可以自行尝试在只保存了 name 属性或只保存了 displayname 时，系统通讯录中显示的名称有什么区别，从而理解手机通讯录中对联系人名称的保存机制，这对真正开发一款好用的通讯录应用很有帮助。

6.4 联系人的创建、读取、修改和删除

在范例 6-3 中利用 Contact 对象创建了一个新的联系人，Contact 除了可以创建一个联系人之外还可以实现对联系人的修改、删除和复制等操作。本节将以一个例子来实现这些功能。

【范例 6-4】联系人的创建、读取、修改和删除。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <script src="cordova.js" type="text/javascript" charset="utf-8"></script>
06  <script type="text/javascript" charset="utf-8">
```

```
07        // 等待加载 Cordova
08        document.addEventListener("deviceready", onDeviceReady, false);
09        // 一般需要在设备加载完毕后再使用 API，以确保 API 能够正常使用
10        // Cordova 加载完毕
11        function onDeviceReady() {
12            //新建一个联系人
13            var contact1 = navigator.contacts.create();
14            contact1.displayName = "Dog";
15            contact1.nickname = "Doggie";       //同时指定以支持所有设备
16            // 填充一些字段
17            contact1.addrsses= "anywhere";
18            var name1 = new ContactName();
19            name1.givenName = "Wang";
20            name1.familyName = "Wang";
21            contact1.name = name1;
22            //保存
23            contact1.save(onSaveSuccess,onSaveError);
24            //再新建一个联系人
25            var contact2 = navigator.contacts.create();
26            contact2.displayName = "Cat";
27            contact2.nickname = "Cattie";           //同时指定以支持所有设备
28            // 填充一些字段
29            contact2.addrsses= "anywhere";
30            var name2 = new ContactName();
31            name2.givenName = "Miao";
32            name2.familyName = "Miao";
33            contact2.name = name2;
34            //保存
35            contact2.save(onSaveSuccess,onSaveError);
36            //克隆一个联系人
37            var clone = contact1.clone();
38            alert(clone.nickname);
39        //删除一个联系人
40            contact1.remove(onRemoveSuccess,onRemoveError);
41        }
42        //新建联系人成功
```

```
43        function onSaveSuccess(contacts){
44            alert("新建联系人成功");
45        }
46        //新建联系人失败
47        function onSaveError(contactError){
48            alert("新建联系人失败");
49        }
50        //删除联系人成功
51        function onRemoveSuccess(contacts){
52            alert("删除联系人成功");
53        }
54       //删除联系人失败
55        function onRemoveError(contactError){
56            alert("删除联系人失败");
57        }
58    </script>
59    </head>
60    <body>
61        <h1>Cordova 的联系人</h1>
62    </body>
63    </html>
```

不直接运行程序，先对代码做一些简单地分析。

首先要做的依然是等待设备加载完毕，如范例第 9 行所示。当设备加载完毕之后，系统会自动执行自定义函数 onDeviceReady。在此处由于确定设备已经被完全加载了，就可以开始创建联系人了。本例首先创建了两个联系人，其中一个的各项资料都是与 Dog 有关的，如第 13~21 行所示，此时只是创建了一个 Contact 对象，并不是真正地将数据写入到通讯录中。因此需要在第 23 行使用 save()方法将数据写入到通讯录中。

接下来又新建了一个联系人，而这个联系人都是与 Cat 有关的，如范例第 25~33 行所示。将新创建的联系人保存为 contact2，然后写入到通讯录中。这两个联系人保存完毕之后都会在回调函数 onSaveSuccess 中弹出对话框来提示联系人保存成功。

第 37 行使用 clone()方法来对 contact1（也就是那个各项资料都与 Dog 有关的联系人）进行了一次复制，将它的所有信息都保存到了名为 clone 的联系人对象中，然后通过 remove()方法对 contact1 进行删除。

这样一来实际上最终通讯录中只剩下那位各种资料都与 Cat 有关的联系人了，那么接下来可以看看通讯录中的结果是不是这样的，如图 6-6 所示。

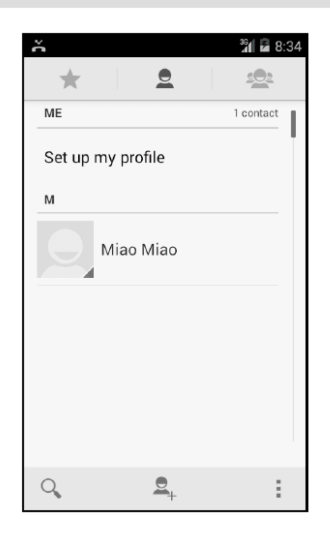


图 6-6　运行程序之后的通讯录

6.5　ContactField 对象的深入研究

在学习 Contact 对象的属性时，有不止一个属性是 ContactField 类型的变量，那么 ContactField 类究竟是什么呢？本节将对它进行比较深入地剖析。

ContactField 对象是一个可重用的组件，用于支持通用方式下的联系人字段，每个 ContactField 对象中都包含了一个值属性（value）、一个类型属性（type）和一个首选项属性（pref）。其中 type 属性是一个字符串，用来对该对象的类型进行说明，比如说它的值可以为“这是一个电子邮件”。value 属性用来存放具体的值，依然以邮件为例，它的值就可以是 123456@sina.com。pref 是一个布尔型变量，它包含了当前用户的首选项，默认为 true。

在大多数情况下，ContactField 对象中的 type 属性并不会是事先确定值。例如，一个电话号码的 type 属性值可以是：home、work、mobile、iPhone 或其他相应特定设备平台的联系人数据库所支持的值。然而对于 Contact 对象的 photos 字段，Cordova 使用 type 字段来表示返回的图像格式。如果 value 属性包含的是一个指向照片图像的 URL，Cordova 对于 type 会返回 url；如果 value 属性包含的是图像的 Base64 编码字符串，Cordova 对于 type 会返回 base64。

范例 6-5 是一个使用 ContactField 对象的例子。

【范例 6-5】使用 ContactField 对象。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <script src="Cordova.js" type="text/javascript" charset="utf-8"></script>
05  <script type="text/javascript" charset="utf-8">
```

```
06      // 等待加载 Cordova
07      document.addEventListener("deviceready", onDeviceReady, false);
08      // Cordova 加载完毕
09      function onDeviceReady() {
10          // 建立一个新的联系人记录
11          var contact = navigator.contacts.create();
12          //存储联系人电话号码到 ContactField[]数组
13          var phoneNumbers = [3];
14          phoneNumbers[0] = new ContactField('work', '212-555-1234', false);
15          phoneNumbers[1] = new ContactField('mobile', '917-555-5432', true);
16          // 首选项
17          phoneNumbers[2] = new ContactField('home', '203-555-7890', false);
18          contact.phoneNumbers = phoneNumbers;
19          // 存储联系人
20          contact.save();
21
22          // 搜索联系人列表，返回符合条件联系人的显示名及电话号码
23          var options = new ContactFindOptions();
24          options.filter="";
25          filter = ["displayName","phoneNumbers"];
26          navigator.contacts.find(filter, onSuccess, onError, options);
27      }
28
29      // onSuccess：返回联系人结果集的快照
30      function onSuccess(contacts) {
31          for (var i=0; i< contacts.length; i++) {
32          // 显示电话号码
33              for (var j=0; j< contacts[i].phoneNumbers.length; j++) {
34                  alert("类型： " + contacts[i].phoneNumbers[j].type + "\n" +
35                      "值： "  + contacts[i].phoneNumbers[j].value + "\n" +
36                      "首选项： "  + contacts[i].phoneNumbers[j].pref);
37              }
38          }
39      }
40
41      // onError：获取联系人结果集失败
42      function onError() {
43          alert('onError!');
44      }
```

```
45  </script>
46  </head>
47  <body>
48      <h1>ContactField 对象的使用</h1>
49  </body>
50  </html>
```

在 Contact 对象中，电话号码（phoneNumbers）就是一个 ContactField 类的数组对象，范例 13 行将电话号码声明成了一个数组。在第 14~17 行为数组中的对象赋值，然后在第 34~36 行通过 alert()函数将它们显示出来，运行结果如图 6-7 所示。

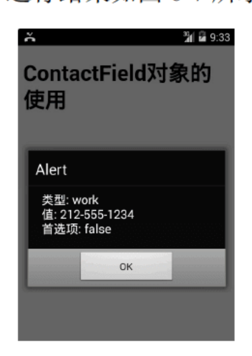


图 6-7　显示电话号码的各个属性值

可以看到，被消息框弹出的内容与最初在程序中输入的内容是对应的，使用这样的方法能够保证在安卓自带通讯录中项目不足时为用户提供额外的数据保存选项。



某些系统中不支持 ContactField 类型的首选项属性，因此该值在安卓、黑莓以及 iOS 设备下永远为 false。事实上笔者认为在安卓通讯录中的许多字段对普通用户来说实在是有些过于烦琐了，虽然这些内容有助于保证满足所有人的需求，但是作为开发者就需要在这方面下一些功夫。这在一定程度上也能够解释为什么在智能机横行的今天，许多用户开始愿意购买一款像诺基亚 1050 这样的功能机来使用。

6.6 小结

本章学习了利用 Cordova 对设备的联系人信息进行修改和查看的方法。通过这些方法可以帮助用户实现一些非常有用的改良。读者要注意在原生安卓通讯录中的一些信息（比如地址、邮编，还有国外用户习惯的将姓和名分开存储的方法）实际上都是不太适合中国人使用习惯的，这时就可以利用本章所介绍的内容来制作一款更适合国内用户的通讯录软件。

第 7 章

Cordova的消息提示

本章将介绍 Cordova 中消息提示的方法，在之前的内容中为了显示一些信息，通常使用 JavaScript 中的 alert()方法或是将信息在 LogCat 中显示。这虽然不失为一种不错的方法，但显然无法满足实际项目中的需要。本章要介绍的 notification 类就能很好地解决这一需求。Cordova 的 notification 类封装了一系列设备视觉、听觉和触觉的通知，可以为应用定制专门的消息推送、警告或提示。本章还将举例介绍 Cordova 中每一种消息通知可以使用的场景，以便使读者能够在实际开发中得心应手。

本章的主要知识点有：

- 利用 notification.alert()弹出一个警告或者对话框。
- 利用 notification.confirm()显示一个可定制的确认对话框。
- 利用 notification.prompt()弹出一个对话框。
- 利用 notification.beep()控制蜂鸣器使设备发出蜂鸣声。
- 利用 notification.vibrate()控制产生一定时长的震动。
- notification 类中每种方法使用的场合以及各种方法可能出现的 bug。

7.1 notification 警告的使用

notification.alert()方法可以使设备弹出一个可以自定义的对话框，之前这样的功能主要依靠 JavaScript 中的 alert()方法来实现。与 alert()方法相比，显然 notification.alert()具有更加强大的可定制性与易用性，范例 7-1 是它的一个例子。

首先需要在命令行执行命令添加插件：cordova plugin add cordova-plugin-dialogs

【范例 7-1】使用 notification.alert()方法显示警告。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <title>使用 notification.alert()方法显示警告</title>
05  <script type="text/javascript" charset="utf-8" src="js/cordova.js">
</script>
06  <script type="text/javascript" charset="utf-8">
```

```
07      // 等待加载 Cordova
08      document.addEventListener("deviceready", onDeviceReady, false);
09      // Cordova 加载完毕
10      function onDeviceReady() {
11          //使用 alert()方法显示一个提示
12          alert("设备加载完毕");
13      }
14      // 警告对话框被忽视
15      function alertDismissed() {
16          // 进行处理
17      }
18      // 显示一个定制的警告框
19      function showAlert() {
20          navigator.notification.alert(
21              '这是使用 notification.alert()方法显示警告',   // 显示信息
22              alertDismissed,                               // 警告被忽视的回调函数
23              '这是一个标题',                                // 标题
24              '我就不说这个按钮是可以自定义的'                  // 按钮名称
25          );
26      }
27  </script>
28  </head>
29  <body>
30      <h1 onclick="showAlert();">显示对话框</h1>
31  </body>
32  </html>
```

运行结果分别如图 7-1 和图 7-2 所示。

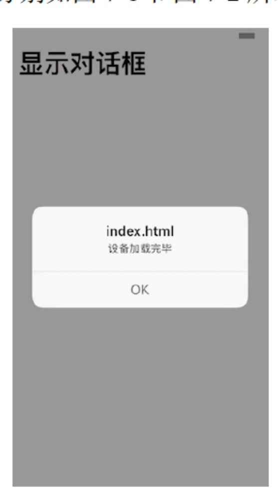


图 7-1　使用 JavaScript 的 alert()方法显示对话框



图 7-2　使用 notification 显示一个自定义的对话框

可以看到，使用两种方法显示出的对话框在样式上有一定的区别，而且使用 notification 类产生的对话框具有更强大的自由度，比如可以自定义对话框的标题、按钮等的内容。除此

之外，观察范例第 20~25 行，很明显 notification 中的 alert()方法还具有方便调用的回调函数，这也是它要比原生 JavaScript 中的 alert()方法更加方便的原因之一。

范例第 20~25 行就是 notification.alert()方法的使用，它有 4 个参数，分别是对话框的内容、回调函数、对话框的标题和对话框按钮的标题。



在大多数平台中，notification.alert()与 alert()方法将会显示为与原生 SDK 相同的对话框样式，但是在少数低版本平台中，如塞班则会显示成浏览器中对话框的样式。如在图 7-1 中使用 alert()方法所显示出的对话框就是浏览器中默认的对话框样式，可以看出与图 7-2 中的对话框有明显的区别。但是在比较新的版本中，无论采用哪种方式显示出的对话框都是一样的。

7.2 notification 确认对话框的使用

除了 alert()之外，notification 类还提供了 confirm()对话框，使用方法与 notification.alert()非常类似，范例 7-2 是使用它弹出对话框的一个例子。

【范例 7-2】使用 confirm()方法弹出对话框。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>使用 notification.confirm()方法显示警告</title>
06  <script type="text/javascript" charset="utf-8" src="js/cordova.js">
</script>
07  <script type="text/javascript" charset="utf-8">
08      // 等待加载 Cordova
09      document.addEventListener("deviceready", onDeviceReady, false);
10      // Cordova 加载完毕
11      function onDeviceReady() {
12          //使用 alert()方法显示一个提示
13          alert("设备加载完毕");
14      }
15      // 警告对话框被忽视
16      function onConfirm() {
17          // 进行处理
18          alert("点击按钮后运行回调函数");
19      }
```

```
20      // 显示一个定制的警告框
21      function showConfirm() {
22          navigator.notification.confirm(
23              '这是使用 notification.confirm()方法显示对话框', // 显示信息
24              onConfirm,                                    // 警告被忽视的回调函数
25              '这是一个标题',                                // 标题
26              '这个按钮也是可以自定义的'                      // 按钮名称
27          );
28      }
29  </script>
30  </head>
31  <body>
32      <h1 onclick="showConfirm();">显示对话框</h1>
33  </body>
34  </html>
```

编译运行之后，点击屏幕上方的“显示对话框”弹出对话框（如图 7-3 所示），当点击对话框中的按钮后运行回调函数 onConfirm()，弹出一个新的对话框如图 7-4 所示。



图 7-3　弹出确认对话框



图 7-4　触发回调函数弹出新的对话框

范例第 22~27 行是 Cordova 中使用 confirm()方法的一个例子，该方法仍然使用了回调函数，但是值得注意的是，在 confirm()方法中，当用户点击了对话框中的按钮后才执行回调函数，这与上一节中的 alert()方法有所区别。

这种方法的使用非常有意义，因为在 Cordova 中默认的对话框只能起到提醒的作用，但是无论用户确认与否系统都会默认按照原定的计划执行。这有一种先斩后奏的味道，很有可能会引起某些用户的反感，如果使用了这种具有确定功能的对话框，则可以避免此类现象的发生。

7.3 notification 显示可以传递变量的对话框

经过前面两节的学习已经了解了在 Cordova 中使用对话框的方法，不过显然它们还不能够满足实际开发中对于对话框的要求。比如经常需要通过对话框来传递一些数值或用来表示用户是确定还是取消的选择结果，这时就需要一种更加强大的对话框。这样的需求可以靠 notification 类中的 prompt()方法来实现。

【范例 7-3】利用 notification.prompt()方法实现更高级的对话框。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>使用 notification.prompt()方法显示对话框</title>
06  <script type="text/javascript" charset="utf-8" src="js/cordova.js">
</script>
07  <script type="text/javascript" charset="utf-8">
08      // 等待加载 Cordova
09      document.addEventListener("deviceready", onDeviceReady, false);
10      // Cordova 加载完毕
11      function onDeviceReady() {
12          //使用 alert()方法显示一个提示
13          alert("设备加载完毕");
14      }
15      //从对话框中获得下一步操作的数据
16      function onPrompt(results) {
17          // results 是从 Prompt 对话框中获取的值，buttonIndex 存放被点击的按钮的索引
18          if(results.buttonIndex==1){
19              //results.input1中存放的是编辑框中输入的值
20              alert("你要了"+results.input1+"杯咖啡");
21          }
22      }
23      // 显示一个定制的对话框
24      function showPrompt() {
25           navigator.notification.prompt(
26              '请问你需要喝杯咖啡么？',                  // 显示信息
27              onPrompt,                                  // 警告被忽视的回调函数
28              '将想要的数量输入在对话框内',              // 标题
29              ['要','不要']                              // 按钮名称
```

```
30            );
31        }
32  </script>
33  </head>
34  <body>
35      <h1 onclick="showPrompt();">显示对话框</h1>
36  </body>
37  </html>
```

编译之后运行结果如图 7-5 所示，在编辑框中输入数字 1，并选择“要”之后，得到如图 7-6 所示的界面，显示用户需要 1 杯咖啡，显然用户的选择确实通过对话框传递给了应用。



图 7-5　输入数字通知系统

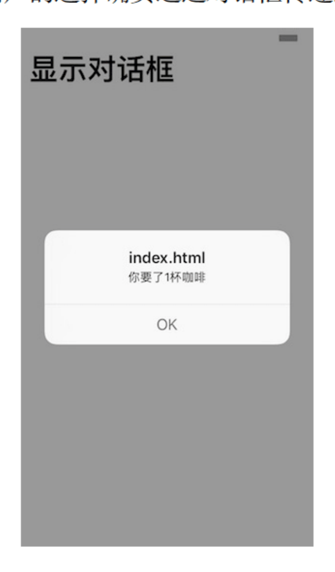


图 7-6　用户输入的数字将会通过对话框传递给系统

范例第 25~30 行是使用 notification.prompt()方法的一个例子，与之前学过的两种对话框也非常相似，但是在第 4 个参数（也就是第 29 行）又与之前有一些区别。在 prompt()方法的第 4 个参数中，用方括号括起对话框按钮的多个标题，并用逗号分开。



虽然在 prompt()方法中可以支持使对话框超过三个按钮选项，但是当标题超过三个时就只会显示前三个标题。

范例第 16 行定义了回调函数 onPrompt()，其中的 result 类封装了对话框中提交的值，results.buttonIndex 中存储的是对话框中各个按键的索引，results.input1 中存放了用户在编辑框中输入的文本。

本节介绍的这种功能看似非常实用，但是由于弹出对话框的样式过于死板，比如仅能支持一个输入栏而不能采用更加适合用户使用的下拉列表等形式，因此在实际开发中常常使用精心制作的 jQuery 插件来实现类似的需求。

7.4 notification 控制蜂鸣器和震动

前面已经介绍了三种使用 notification 类创建和使用对话框的方法，除了利用对话框从视觉上对用户进行提示之外，还可以依靠 beep()方法和 vibrate()方法操纵设备发出蜂鸣声和震动。范例 7-4 是使用这两种方法的一个例子。

【范例 7-4】使用 notification 类操纵蜂鸣器和震动。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>使用 notification 类操纵蜂鸣器和震动</title>
06  <script type="text/javascript" charset="utf-8"
src="js/cordova.js"></script>
07  <script type="text/javascript" charset="utf-8">
08      // 等待加载 Cordova
09      document.addEventListener("deviceready", onDeviceReady, false);
10      // Cordova 加载完毕
11      function onDeviceReady() {
12          //使用 alert()方法显示一个提示
13          alert("设备加载完毕");
14      }
15      //显示对话框
16      function showAlert() {
17         navigator.notification.alert(
18             '这是一个对话框',           // message
19             '这是对话框的标题',         // title
20             '确定'                      // buttonName
21         );
22      }
23      //蜂鸣器响3次
24      function playBeep() {
25         navigator.notification.beep(3);
26      }
27      //连续震动两分钟
28      function vibrate() {
29         navigator.notification.vibrate(2000);
30      }
31  }
32  </script>
33  </head>
34  <body>
```

```
35        <h1 onclick="showAlert();">显示对话框</h1>
36        <h1 onclick="playBeep();">蜂鸣器发声</h1>
37        <h1 onclick="vibrate();">使设备震动</h1>
38    </body>
39    </html>
```



在最新版本的 cordova 项目中如果需要调用 navigator.notification.vibrate()需要在项目中添加插件 cordova-plugin-vibration。

运行后分别点击“显示对话框”“蜂鸣器发声”“使设备震动”可以分别弹出对话框、使蜂鸣器发出三次声音、使设备震动 2s。

第 25 行是 Cordova 中利用 notification.beep()方法调用蜂鸣器的例子，其中的参数代表了蜂鸣器发出声音的次数，第 29 行利用 notification.vibrate()方法使设备震动，其中的参数是使设备连续震动的时间，单位为 ms。

至此 notification 类中的各种方法就介绍完了，但这仅仅是开始，最重要的还是理解它们之间的优劣关系，以便能够在合适的地方使用合适的方法来达到完美的用户体验。

首先将各种对话框归为一类，由于当对话框被弹出时，应用将被暂时置为不可用，会对用户的使用造成打断的效果，且用户不得不对该提示做出回应，因此对话框一般用来作为一些重要信息的提示使用。比如游戏中可以在系统电量较低时使用对话框来提醒用户及时充电，或者在用户要退出应用时来向用户确认是否为误操作。另外，在用户执行一些会影响到系统的操作（比如删除文件、删除联系人等）时也需要弹出对话框来让用户确认。

由于蜂鸣器和震动可以被用户主动忽略，因此主要用来增强应用的交互性，比如一款新闻浏览应用可以在有新闻时利用蜂鸣器来提醒用户，而一些小游戏也可以利用震动来增强用户的游戏体验，比如当用户操纵的人物死亡时设备震动表示处罚。蜂鸣器和震动也可以与对话框搭配使用。

蜂鸣器与震动之间也有一定的区别，蜂鸣器的声音过于单调，因此在许多环境下难以适应用户体验的需求，比如用户点击屏幕时可以采用一个几毫秒的震动来作为回馈，但是如果利用蜂鸣器来作为反馈就会让用户感到心烦，因此在实际开发中一定要慎重选择。

7.5 小结

本章主要学习了利用 notification 类中的各种方法来实现对用户的各种反馈，其中包括了对话框、蜂鸣器和震动。这些方法对提升一款应用的交互性会起到非常重要的作用。除此之外，这些反馈的方法还可以使开发者更好地实现对应用的调试，比如可以利用 prompt()来中断程序并修改某个变量的值，这些都极大地方便了 Cordova 开发人员。

第 8 章

加速度传感器

本章将介绍 Cordova 中使用加速度传感器获取数据的方法。加速度传感器是一种用来检测物体相对运动方向以及沿着 x、y、z 三个方向运动的部件，现代的智能手机绝大多数都集成了加速度传感器。Cordova 提供了 accelerometer 类用来接收来自加速度传感器的数据，从而可以检测到设备在空间上的位置变化。

本章的主要知识点有：

- 如何获取设备在各个方向上的加速度。
- 如何按照一定的时间间隔获取设备在各个方向上的加速度。
- 在特定的时间获取设备在各个方向上的加速度。
- 如何处理获取到的加速度数据。

8.1 获取当前的加速度

在 Cordova 中，类 accelerometer 封装了专门用来获取设备加速度的方法，其中最基础的一个方法就是 getCurrentAcceleration()。accelerometer.getCurrentAcceleration()的作用是获得设备当前（即该方法被调用时）的相对运动方向，范例 8-1 展示了该方法的使用。

首先在命令行执行命令添加插件：cordova plugin add cordova-plugin-device-motion。

【范例 8-1】使用 accelerometer.getCurrentAcceleration()获取设备当前运动信息。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>使用 accelerometer.getCurrentAcceleration()获取设备当前运动信息</title>
06  <script type="text/javascript" charset="utf-8"
src="js/cordova.js"></script>
07  <script type="text/javascript" charset="utf-8">
08      // 等待加载 Cordova
09      document.addEventListener("deviceready", onDeviceReady, false);
10      // Cordova 加载完毕
11      function onDeviceReady() {
12          //使用 alert()方法显示一个提示
13          alert("设备加载完毕");
```

```
14        }
15        function getDevAccelecation() {
16            //获取设备加速度信息
17            navigator.accelerometer.getCurrentAcceleration(onSuccess, onError);
18        }
19        function onSuccess(acceleration) {
20            //将获取的信息显示出来
21            alert('Acceleration X: ' + acceleration.x + '\n' +
22                    'Acceleration Y: ' + acceleration.y + '\n' +
23                    'Acceleration Z: ' + acceleration.z + '\n' +
24                    'Timestamp: ' + acceleration.timestamp + '\n');
25        }
26        function onError() {
27            //若获取设备信息失败则显示 onError
28            alert('onError!');
29        }
30    </script>
31    </head>
32    <body>
33        <h1 onclick="getDevAccelecation();">获取设备运动信息</h1>
34    </body>
35    </html>
```

运行后点击屏幕上的“获取设备运动信息”字样，可以弹出如图 8-1 所示的对话框，分别显示设备在 x、y、z 三个方向上的加速度以及测试加速度的时间戳。



图 8-1　显示当前设备的加速度信息



许多虚拟机不支持加速度传感器，因此会弹出显示 onError 的对话框，因此要尽量在真机上进行调试。如图 8-2 就是笔者在虚拟机上运行的结果。

范例第 17 行是系统使用 accelerometer.getCurrentAcceleration()方法获取设备的当前运动信息，它有两个参数，分别是两个自定义函数的函数名。第一个函数在获取信息成功后被调用，同时还会接收到一个 acceleration 对象，此对象中封装了与加速度有关的信息，分别是设备在各个方向上的加速度以及当前的时间戳。第二个函数在获取设备信息失败时被调用，比

如第 26~29 行的 onError()函数就用来反馈获取设备加速度信息失败的消息。



图 8-2　在虚拟机中弹出失败错误信息

8.2 监视设备的加速度

上一节学习了利用 getCurrentAcceleration()方法获取设备当前加速度的方法。也许有读者已经想到了它的一些作用，比如可以用来记录用户每天的运动量，但这就遇到了一个问题。如果要记录用户每天的运动量就需要不断地调用 getCurrentAcceleration()方法来进行计算，虽然可行，但这无疑十分麻烦。有没有更好的方法呢？

Cordova 作为一款比较完善的跨平台开发框架自然有过这方面的考虑，封装在类 accelerometer 中的 watchAcceleration() 无疑比利用 JavaScript 不断去调用 getCurrentAcceleration()要方便得多。范例 8-2 是使用 watchAcceleration()方法监视手机加速度变化的一个例子。

【范例 8-2】利用 accelerometer.watchAcceleration()方法监视设备的加速度变化。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>利用 accelerometer.watchAcceleration()方法监视设备的加速度变化</title>
06  <script type="text/javascript" charset="utf-8" src="js/cordova.js">
</script>
07  <script type="text/javascript" charset="utf-8">
08      // 等待加载 Cordova
09      document.addEventListener("deviceready", onDeviceReady, false);
10      // Cordova 加载完毕
```

```
11       function onDeviceReady() {
12           //使用 alert()方法显示一个提示
13           alert("设备加载完毕");
14       }
15       //监视设备加速度
16       function startWatch() {
17           //每0.3秒更新一次加速度信息
18           var options = { frequency: 300 };
19           watchID = navigator.accelerometer.watchAcceleration(onSuccess,
onError,options);
20       }
21       //停止监视设备加速度
22       function stopWatch() {
23           if (watchID) {
24               navigator.accelerometer.clearWatch(watchID);
25               watchID = null;
26           }
27       }
28       //获取加速度信息成功
29       function onSuccess(acceleration) {
30           //处理获取的加速度信息
31           str_x = 'x: ' + acceleration.x;                //x 方向的加速度
32           str_y = 'y: ' + acceleration.y;                //y 方向的加速度
33           str_z = 'z: ' + acceleration.z;                //z 方向的加速度
34           str_t = 'Time: ' + acceleration.timestamp;     //获取加速度的时间戳
35           //将处理后的数据显示到屏幕上
36   document.getElementById("info").appendChild(document.createTextNode(str_x));
37   document.getElementById("info").appendChild(document.createTextNode(str_y));
38   document.getElementById("info").appendChild(document.createTextNode(str_z));
39   document.getElementById("info").appendChild(document.createTextNode(str_t));
40       }
41       function onError() {
42           //若获取设备信息失败则显示 onError
43           alert('onError!');
44       }
45   </script>
46   </head>
47   <body id="info" bgcolor="#E4E4E4">
48       <h1 onclick="startWatch()">获取设备运动信息</h1>
```

```
49        <h1 onclick="stopWatch()">停止获取设备运动信息</h1>
50    </body>
51    </html>
```

运行之后点击屏幕上的“获取设备运动信息”字样，然后用手不断地晃动手机使之发生位移，可以看到屏幕上的数据不断变化。获取一些加速度信息后再点击“停止获取设备运动信息”的字样，屏幕中的数据将不再变化。运行后结果如图 8-3 所示。



图 8-3　监视设备的加速度变化



由于本节重点讲解基础知识，不涉及 CSS 样式问题，因此界面会比较混乱。

阅读范例中的代码，第 16~20 行声明了一个函数 startWatch()，第 19 行使用了 watchAcceleration()方法来监视设备的加速度变化情况。观察该函数可以发现，除了 onSuccess 和 onError 之外，它比 getCurrentAcceleration()方法还多了一个参数 option，该参数用来设置每次监控设备加速度信息的时间间隔，以毫秒为单位，本范例中就设置为每 0.3s 获取一次加速度信息（第 18 行）。

在 onSuccess 函数中，acceleration 类的用法与上面介绍的完全一样，但在 36~39 行中使用了 JavaScript 中的 DOM 方法将获取到的信息显示到页面上。

除此之外可以看到，在本例中还多了一个自定义的 stopWatch()方法，它的作用是调用 clearWatch()方法来停止对设备加速的监听，这样就保证了用户可以决定在什么时候开始和停止监听设备的运动情况。依然拿记录用户的运动量来举例。如果没有 clearWatch()方法，那么手机将会记录任何时刻用户的总运动量，但是有了 clearWatch()方法之后，用户就可以自行决定在什么时候要对自己的运动量进行计算，比如可以计算用户在一个小时内的运动量。

另外观察图 8-3 中 Time 值的变化情况，可以发现每显示一条数据，时间戳增大的数字大约为 10000，可见该时间戳记录的单位为毫秒（ms），用来记录设备已经使用的总时间。


可以暂停监控一段时间再重新开始监控，然后观察时间戳的变化来验证。有兴趣的读者也可以根据该值来计算笔者写本节内容所花费的时间。

8.3　详解 acceleration 对象

在之前的例子中曾经多次使用到 acceleration 对象，可是 acceleration 对象究竟是什么呢？相信不少读者并没有真正明白。本节将深入剖析 acceleration 对象以及它的各个属性。

acceleration 对象中封装了由 watchAcceleration()方法或 getCurrentAcceleration()方法获取的数据，其中包括了 4 个属性，分别是 x、y、z 和 timestamp。其中 x、y、z 分别保存了设备在 x、y、z 三个方向上的加速度，单位是米/二次方秒（m/s^2）。另外，由于设备还受到重力的影响，所以当设备被安静地放在某个地方静止不动时，获取的加速度应为 x=0、y=0 和 z=9.81，其中 9.81 为地表的重力加速度。


这里要介绍一下平时常说的重力感应功能是怎样实现的。由于当手机平放静止时，仅有 z 方向受到重力，而 x、y 方向的重力加速度均为 0。但是当手机发生倾斜时，就变成了由三个方向合成重力加速度，即 x、y、z 三个方向的平方和为 9.81 的平方。通过测量每个方向的重力加速度就可以得到手机倾斜的角度了。

另外，由于设备获取的往往是一瞬间的加速，因此在使用获取的加速数据时还需要对这些数据进行一些处理，以免操作过于突兀。比如读者一定都遇到过这样的问题，测量一包大米的重量，为了保证精确需要测量多次求平均值。但是假如测量了 4 次的数据分别是 0.5kg、0.5kg、0.55kg 和 100 斤，那么显然 100 斤是错误的。所以最终要对 0.5kg、0.5kg 和 0.55kg 求平均值。在实际开发中也是这样，假如 x 方向以 100ms 为间隔监控 x 方向的加速度分别为 1、1、1、↕0.5、1，那么就需要考虑一下这个↕0.5 要怎么处理了。

8.4　加速度传感器的使用

前面几节已经介绍了获取设备加速度信息的方法，本节再来介绍一些加速度传感器适用的场合。

8.4.1　游戏

这个应该是首先能想到的了，像比较经典的极品飞车 17 就采用了加速度传感器（如图 8-4 所示），利用手机左右摇摆产生的加速度来控制车辆前行的方向。另外还有一些飞行射击游戏也是利用了类似的方法，比如《钢铁侠》系列游戏就是利用手机的左右摇摆来控制人

物躲避迎面飞来的武器，还有一些跑酷游戏（如神庙逃亡）也是如此。

另外一类游戏就是迷宫，可以利用手机的倾斜来控制小球四处转动最终逃出迷宫。由此可见加速度传感器应用在游戏中一般用来操纵主角进行移动，这样也比较符合用户的思维习惯。



相信不少读者都玩过老式的手柄游戏，在主角进行跳跃时不少人的身体都会随着跳跃的方向进行倾斜也是类似的道理。



图 8-4　极品飞车 17 的游戏画面

8.4.2　抽奖

加速度传感器的作用就是获取设备的空间变化信息，那么设备的摇动自然会有足够的数据变化，而晃动手机的动作也容易让用户联想到抽签或者抓阄时的忐忑心情，因此一些应用采用了加速度传感器来模拟用户抽奖的行为。此外微信等社交应用的“摇一摇”功能也是模拟了用户的这一思维习惯。

8.4.3　更多更强大的交互

实际上仍然是依靠加速度传感器的变化来实现对设备的操纵，但是范围要更广一些。比如一些手机防丢的应用会在加速度较大的时候发出警报，或者是在用户握住手机做甩鞭子动作时，发出鞭子抽打身体的声音，这些都是使用加速度传感器的例子。

8.5　实战：制作“马上有一切”的动画

以上介绍了加速度传感器可以使用的场合，本节将以一个简单的例子来实现一款“拜年”的应用：马上有一切。

2014 年是马年，于是马这种生物一下子就变成了人民群众喜闻乐见的吉祥物，无所不能的网友们借着这一契机，发明了许多有意思的图画，比如“马上有房”（见图 8-5）、“马上有车”，甚至是“马上有对象”（见图 8-6）等。笔者就有了一个想法，可以此为题材做

一款拜年的手机应用。



图 8-5 “马上有房子”图片



图 8-6 “马上有对象”图片

8.5.1 原形设计

笔者希望在手机屏幕上显示出一个卡通马的造型，马背朝向屏幕的右上方，马背顶部为空白。当用户摇晃手机时，马背上方的空白随机显示出房子、钱、汽车图案。为了衬托过年的喜庆气氛还要将背景改为红色，如图 8-7 所示。



图 8-7 界面设计

最初，屏幕上只显示中间的卡通马图案，当用户摇晃手机后，随机以渐显的形式显示出钱、房子或汽车的图案，再在屏幕的下方显示相应的文字：“马上有钱/房/车”。

8.5.2 素材准备

在理清思路之后就开始准备素材了，本项目需要 4 张图片，内容依次为马、钱、汽车、房子，如图 8-8~图 8-11 所示。



图 8-8　素材“马”



图 8-9　素材“钱”



图 8-10　素材“汽车”



图 8-11　素材“房子”



钱、汽车和房子的素材笔者只截取了整个素材的一部分，实际上它们的素材图片尺寸与素材“马”的图片尺寸是相同的，这样摆它们和马之间的位置保持背景透明。只要让它们的位置重叠就可以实现在马背上的效果了。

8.5.3　动画实现

在 Cordova 中新建一个文件夹，命名为 images，其中放置这 4 张图片，目录如图 8-12 所示。

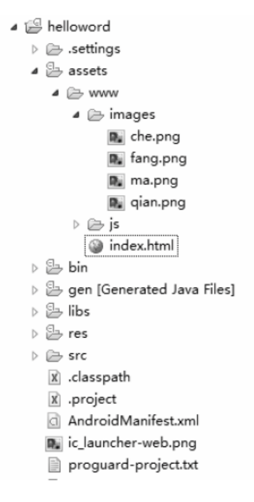


图 8-12　新建的图片目录

使用生成 1~3 的随机数来决定以渐显的方式显示某一图片的代码，如范例 8-3 所示。

【范例 8-3】随机显示图片。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>马上有一切</title>
06  <script type="text/javascript" charset="utf-8" src="js/cordova.js">
</script>
07  <script type="text/javascript" charset="utf-8">
08      //初始化图片的透明度，并生成一个1~3的随机数
09      var iopa = 0;
10      randNum=Math.floor(Math.random()*3);
11      // 等待加载 Cordova
12      document.addEventListener("deviceready", onDeviceReady, false);
13      // Cordova 加载完毕
14      function onDeviceReady() {
15          //根据随机数加载3幅素材文件，获取要显示的贺词
16          pic = new Array("qian.png","che.png","fang.png");
17          name = new Array("钱","车","房");
18          //利用随机数获取要显示的图片资源，将 img 样式图片赋给 id
19          document.getElementById("img").style.background = "url(images/"+
                pic[randNum]+")";
20      document.getElementById("heci").appendChild(document.createTextNode("马
上有"
21      +name[randNum]));
22          //开始图像渐显效果
23          start();
24      }
25      function start() {
26          //设置一个计数器
27          var iID=setInterval(opa_change, 50);
28      }
29      function opa_change() {
30          if(iopa==1)
31          {
32              //当透明度为1，图片显示完全，计数器结束
33              clearInterval(iID);
34          }else
```

```
35          {
36              //透明度降低
37              iopa=iopa + 0.02;
38              document.getElementById("img").style.opacity = iopa;
39          }
40      }
41  </script>
42  <style>
43  /*背景颜色为喜庆的大红*/
44  body
45  {
46      margin:0 auto;
47      padding:0;
48      background:#f00;
49  }
50  /*贺词的颜色*/
51  h1
52  {
53      width:100%;
54      text-align:center;
55      color:yellow;
56  }
57  /*显示"马"的图案*/
58  .rec
59  {
60      width:400px;
61      height:480px;
62      background-image:url(images/ma.png);
63      margin-left:40px;
64      margin-top:80px;
65  }
66  /*显示钱、房、车图案的 CSS 样式*/
67  .img
68  {
69      width:400px;
70      height:480px;
71      opacity:0;
72  }
73  </style>
```

```
74 </head>
75 <body>
76     <div class="rec" id="rec">
77         <div class="img" id="img"></div>
78     </div>
79     <h1 id="heci"></h1>
80 </body>
81 </html>
```

编译之后运行结果如图 8-13 所示，马背上渐渐出现了一个大大的“金币”，并在底部显示金黄色的大字“马上有钱”。



图 8-13　屏幕上显示“马上有钱”

范例第 76~78 行用来显示屏幕上图片的页面元素，其中 id 为 rec 的元素用来显示底部的卡通马造型。在其上方覆盖了一块与它大小完全相同的元素，id 为 img 的元素用来显示钱、房子或者车的图案。在准备素材时，不管是马还是其他图案，背景都是透明的，所以相互遮挡之后就能够形成有东西放置在马背上的感觉了。

之后要做的就是让图片以渐显的形式显示了，虽然 jQuery 中有现成的渐显渐隐函数可以使用，但是本范例还是使用原生的 JavaScript。通过 setInterval()函数控制图片的透明度达到渐显的目的（如范例第 27 行所示）。

setInterval()以 50ms 为间隔调用了自定义函数 opa_change()来实现对图片透明度的控制（范例第 29~40 行）。其中变量 ipoa 用来存放图片透明度的具体数值，当该值为 1 时图像完全显示，系统调用函数 clearInterval()来结束计数器的使用。为了配合这一过程在设定 img 元素的样式时就已经将它的透明度设置为 0，如范例第 71 行所示。



本例在第 9 行初始化了变量 ipoa 的值为 0。由于变量作用域的限制，如果在函数 onDeviceReady()中使用将会导致 ipoa 被系统判定为局部变量而无法使用。因此在定义全局变量时一定不要将它声明在函数中。另外，由于 CSS 样式的继承效果，如果给 rec 元素加入透明属性，那么其内部的 img 元素也会跟着透明，因此该属性只能赋给处于顶层的 img 元素使用。

范例第 10 行生成了一个大小为 0~2 的随机数，用来确定最终显示哪幅图片，该值会在onDeviceReady()函数中被使用（范例 16~21 行）。除此之外，还要在屏幕下方显示一行贺词，该功能也是在此处实现的。

8.5.4 最终实现

以上已经实现了“拜年”的动画，本小节要将这一动画与本章学习的加速度传感器结合起来，使它能够给用户一种在庙会抽签祈福的紧张气氛。

【范例 8-4】在应用中加入交互。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>马上有一切</title>
06    <script type="text/javascript" charset="utf-8" src="js/cordova.js">
</script>
07  <script type="text/javascript" charset="utf-8">
08      //初始化图片的透明度，并生成一个0~2的随机数
09      var iopa = 0;
10      //对用户的操作计数
11      var all_roll = 0;
12      var u_roll = 0;
13      randNum=Math.floor(Math.random()*3);
14      // 等待加载 Cordova
15      document.addEventListener("deviceready", onDeviceReady, false);
16      // Cordova 加载完毕
17      function onDeviceReady() {
18          //根据随机数加载3幅素材文件，获取要显示的贺词元素
19          pic = new Array("qian.png","che.png","fang.png");
20          name = new Array("钱","车","房");
21          //利用随机数获取要显示的图片资源，将图片赋给 id 为 img 的样式
22          document.getElementById("img").style.background = "url(images/"+
pic[randNum]+")";
23      document.getElementById("heci").appendChild(document.createTextNode("马
上有"
24      +name[randNum]));
25          alert("摇晃手机之后出现祝福");
26          //开始监测加速度传感器
27          startWatch();
```

```
28        }
29        function start() {
30            //设置一个计数器
31            var iID=setInterval(opa_change, 50);
32        }
33        function opa_change() {
34            if(iopa==1)
35            {
36                //当透明度为1，图片显示完全，计数器结束
37                clearInterval(iID);
38            }else
39            {
40                //透明度降低
41                iopa=iopa + 0.02;
42                document.getElementById("img").style.opacity = iopa;
43            }
44        }
45        //停止监视设备加速度
46        function stopWatch() {
47            if (watchID) {
48                navigator.accelerometer.clearWatch(watchID);
49                watchID = null;
50            }
51        }
52        //监视设备加速度，频率可自行根据需要进行更改
53        function startWatch() {
54            //每0.3s 更新一次加速度信息
55            var options = { frequency: 300 };
56            watchID = navigator.accelerometer.watchAcceleration(onSuccess,
onError,options);
57        }
58        //获取加速度信息成功，判断是否为用户摇晃
59        function onSuccess(acceleration) {
60            //处理获取的加速度信息
61            var pow_v = pow(acceleration.x,2)
62                        + pow(acceleration.y,2)
63                        + pow(acceleration.z,2);
64            var v = sqrt(pow_v);
65            if (v > 15)
```

```
66          {
67              //判断摇晃强度，全部记录和有效记录增加，并轻微震动作为反馈
68              all_roll++;
69              u_roll++;
70              navigator.notification.vibrate(300);
71              //判定为用户晃动，则开始显示图片
72              if(u_roll >10 )
73              {
74                  start();
75              }
76          }else
77          {
78              all_roll++;
79          }
80      }
81      function onError() {
82          //若获取设备信息失败则显示 onError
83          alert('onError!');
84      }
85  </script>
86  <style>
87  /*背景颜色为喜庆的大红*/
88  body
89  {
90      margin:0 auto;
91      padding:0;
92      background:#f00;
93  }
94  /*贺词的颜色*/
95  h1
96  {
97      width:100%;
98      text-align:center;
99      color:yellow;
100 }
101 /*显示"马"的图案*/
102 .rec
103 {
104     width:400px;
```

```
105     height:480px;
106     background-image:url(images/ma.png);
107     margin-left:40px;
108     margin-top:80px;
109 }
110 /*显示钱、房、车图案的 CSS 样式*/
111 .img
112 {
113     width:400px;
114     height:480px;
115     opacity:0;
116 }
117 </style>
118 </head>
119 <body>
120     <div class="rec" id="rec">
121         <div class="img" id="img"></div>
122     </div>
123     <h1 id="heci"></h1>
124 </body>
125 </html>
```

将该范例编译运行之后屏幕上将会弹出对话框提示：“摇晃手机后出现祝福”，之后用力摇晃手机，图像以及贺词将会随着手机的轻微震动渐渐浮现。

首先，此次修改去掉了范例 8-3 中第 23 行处对 start()函数的调用，但是增加了对加速度传感器的监听，如范例第 53~57 行所示。这里就遇到了一个难题，因为即使是一般静止情况下，手机还是会带有一定加速度的，那么怎样才能确定手机是被用户晃动了而非一般的移动或者根本只是正常的轻微幅度抖动呢？

范例第 65 行有一个判断，将来自三个方向的加速度的平方和再开方，得到一个“平均的”加速度。这里将判定是否是人为“摇晃”的临界点定为 15，即当总的加速度大于 15 时判定是人为地摇晃手机。为了更加保险，笔者还做了进一步的约束，在范例 72 行中使用了一个变量 u_roll 记录的大于 15 的晃动次数与 10 次作比较。也就是说，只有获取到 10 次比较剧烈的晃动后系统才会判定用户晃动了手机，开始显示祝贺。



也可以利用每一次晃动略微降低图片的透明度达到图片随着用户的摇晃而变得清晰的效果，在此就不过多赘述了。

另外，为了让用户获得更加真实的体验，在范例第 70 行还使用了震动来作为反馈。注意：本例中震动的时间为 300ms，正好与加速度监听器的频率是相同的，不至于产生两次震动叠加的 bug。

8.6 小结

本章学习了使用 accelerometer 对象获取设备加速信息的方法，并详细地介绍了加速度传感器使用的场合，希望对读者有所启发。此外，本章最后的例子已经开始接触到多种操作互相交互的方法（震动效果与加速度传感器），这也是手机应用交互性的一个趋势。在实际开发中能够使用加速度传感器的地方非常之多，这需要读者在日常生活中不断去积累。

第 9 章

◀ 设备传感器 ▶

上一章介绍了 Cordova 中加速度传感器的调用方法，除了加速度传感器外，Cordova 也提供了对其他设备传感器的调用方法。对设备传感器的使用主要分为对地理位置的获取和方向的获取。为了能够方便开发者调用，Cordova 封装了用来访问 GPS 信息的 Geolocation 类和用来访问设备方向的 Compass 类。

本章的主要内容有：

- 使用 getCurrentPosition()方法获取设备在当前所处的坐标位置。
- 利用 watchPosition()方法对设备所处的坐标位置进行监听。
- 使用 getCurrentHeading()方法获取设备当前的朝向。
- 利用 watchHeading()方法监听设备的朝向。

9.1 利用 Geolocation 类获取设备地理信息

GPS 是现代智能手机中一个非常重要的功能，能够实现全球范围内对设备的实时定位，近几年来在一些社交应用中也经常见到它发挥作用。比如微信的“摇一摇”功能就利用了 GPS 来获取用户的地理位置。图 9-1 是微信利用 GPS 获取两用户之间距离的一个例子。



图 9-1　微信通过 GPS 计算用户之间的距离

Cordova 利用类 Geolocation 来获取设备所在地理位置的坐标。对于一些不支持 GPS 的设备，Geolocation 类还可以借助 IP 地址、RFID、Wi-Fi、蓝牙的 MAC 地址或 GSM/CDMA 手机 ID 的网络信号做出推断，以保证其兼容性。



虽然基于 IP 地址、GSM 基站等方法都可以保证返回设备所在地理位置的经度和纬度，但是这样得到的结果却不一定准确。在实际开发时仍要充分考虑到不支持 GPS 的设备可能存在的误差。

9.2 利用 getCurrentPosition()方法获取设备所在坐标

在 Geolocation 类中，最基本的方法就是 getCurrentPosition()了，它的作用是通过一个 Positon 对象来返回设备所在的位置信息。范例 9-1 是使用 getCurrentPosition()方法的一个例子。

首先在命令行执行命令添加插件:cordova plugin add cordova-plugin-geolocation。

【范例 9-1】使用 getCurrentPosition()方法获取设备位置信息。

```
01 <!DOCTYPE html>
02 <html>
03 <head>
04 <meta charset="utf-8">
05 <title>使用 getCurrentPosition()方法获取设备位置信息</title>
06 <script src="js/cordova.js" type="text/javascript"></script>
07 <script>
08     document.addEventListener("deviceready", onDeviceReady, false);
09     // 设备加载完毕，开始获取位置信息
10     function onDeviceReady() {
11         // 使用 getCurrentPosition()获取位置信息
12         navigator.geolocation.getCurrentPosition(onSuccess, onError);
13     }
14     function onSuccess(position) {
15         // 逐个在屏幕上显示出经度、纬度、海拔等信息
16         document.getElementById("Longitude").innerHTML = "经度: "
+ position.coords.longitude;
17         document.getElementById("Latitude").innerHTML = "纬度: "
+ position.coords.latitude;
```

```
18          document.getElementById("Altitude").innerHTML = "海拔高度："
+ position.coords.altitude;
19          document.getElementById("Accuracy").innerHTML = "精度："
+ position.coords.accuracy;
20          document.getElementById("Altitude Accuracy").innerHTML = "海拔准确
度："
+ position.coords.altitudeAccuracy;
21          document.getElementById("Heading").innerHTML = "航向："
+ position.coords.heading;
22          document.getElementById("Speed").innerHTML = "速度："
+ position.coords.speed;
23          document.getElementById("Timestamp").innerHTML = "时间："
+ position.timestamp;
24      }
25      function onError(error) {
26          alert("获取位置信息失败");
27      }
28  </script>
29  <style>
30  .line                              /* 此样式用于在屏幕上显示一条分割线 */
31  {
32      width:100%;
33      height:1px;
34      border:1px solid #000;
35  }
36  </style>
37  </head>
38  <body>
39      <h1>获取当前位置信息</h1>
40      <div class="line"></div>
41      <h2>你的当前位置：</h2>
42      <h4 id="Longitude">经度：</h4>
43      <h4 id="Latitude">纬度：</h4>
44      <h4 id="Altitude">海拔高度：</h4>
45      <h4 id="Accuracy">精度：</h4>
46      <h4 id="Altitude_Accuracy">海拔准确度：</h4>
47      <h4 id="Heading">航向：</h4>
48      <h4 id="Speed">速度：</h4>
49      <h4 id="Timestamp">时间：</h4>
```

```
50  </body>
51  </html>
```

运行后设备上将显示出当前所处的位置信息，如图 9-2 所示。



图 9-2　使用 getCurrentPosition()获取当前的位置信息

本范例使用了方法 getCurrentPosition()来获取设备所处的位置信息，如第 12 行所示。与其他 Cordova 所提供的 API 相似，该方法同样是将处理成功和失败的两个方法 onSuccess 和 onError 作为参数。

不过在 onSuccess()方法中默认会传送一个 position 对象，该对象保存了两个子对象 coords 和 timestamp。其中 coords 封装了一系列与设备当前所处地理位置有关的信息，比如经度、纬度、海拔等。timestamp 对象则只是单纯地记录了 coords 对象被创建的时间。表 9-1 中是 coords 对象中所包含的各个属性以及它们的含义。

表 9-1　coords 对象中封装的属性

名称	类型	说明
latitude	数字	以十进制小数表示的纬度信息
longitude	数字	以十进制小数表示的经度信息
altitude	数字	海拔高度，单位为米
accuracy	数字	经度和纬度的精确度
altitudeAccuracy	数字	海拔高度的精确度
heading	数字	当前设备运动状态，以正北为零点显示当前运动方向与正北方向的角度
speed	数字	以米/秒为单位显示当前设备运动的速度

第 16~23 行可以看到这些属性的使用，需要注意的是 timestamp 并不只是单纯的以小时来记录 coords 被创建的时间，其中还包含了年、月以及所处的时区等信息。



在 Cordova 中并不是单纯地使用从 GPS 中获取的数据来显示位置信息的，当 GPS 不可用时，Cordova 还会利用网络甚至是与基站的通信来判断当前的位置。因此即使在不具备 GPS 功能的设备上也可以使用该功能。

9.3 使用 watchPosition()方法监控设备的位置变化

本节将介绍一种与上一节中的 getCurrentPosition()方法具有类似功能的方法，只不过它能够对设备的位置信息进行连续监控。看到这一点不知道读者有没有联想到什么？没错，那就是在上一章介绍过的加速度传感器，也有着类似的方法用来监视设备加速度的变化。范例 9-2 中展示了 watchPosition()方法的使用。

【范例 9-2】使用 watchPosition()方法监视设备的位置变化。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta charset="utf-8">
05  <title>使用 watchPosition()方法监视设备的位置变化</title>
06  <script src="js/cordova.js" type="text/javascript"></script>
07  <script>
08      var watchID =null;
09      document.addEventListener("deviceready", onDeviceReady, false);
10      function onDeviceReady() {
11          // 设置监听设备位置信息的频率，单位为毫秒
12          var options = { frequency: 1000 };
13          // 开始监听
14         watchID = navigator.geolocation.watchPosition(onSuccess, onError,
          options);
15      }
16      // 该方法与 getCurrentPosition()方法中所调用的 onSuccess 函数完全一样
17      function onSuccess(position) {
18         document.getElementById("Longitude").innerHTML = "经度: " + position.
           coords.longitude;
19          document.getElementById("Latitude").innerHTML = "纬度: " + position.
           coords.latitude;
```

```
20          document.getElementById("Altitude").innerHTML = "海拔高度: " +
position.coords.altitude;
21          document.getElementById("Accuracy").innerHTML = "精度: " + position.
            coords.accuracy;
22          document.getElementById("Altitude Accuracy").innerHTML = "海拔准确
度: " + position.coords.altitudeAccuracy;
23          document.getElementById("Heading").innerHTML = "航向: " + position.
coords.heading;
24          document.getElementById("Speed").innerHTML = "速度: " +
position.coords.speed;
25         document.getElementById("Timestamp").innerHTML = "时间: " + position.
timestamp;
26      }
27      // 当获取位置信息失败时弹出对话框
28      function onError(error) {
29         alert("获取位置信息失败");
30      }
31      // 调用该函数可以停止对设备位置信息的监听
32      function stop_watch() {
33          navigator.geolocation.clearWatch(watchID);
34      }
35  </script>
36  <style>
37  .line
38  {
39      width:100%;
40      height:1px;
41      border:1px solid #000;
42  }
43  </style>
44  </head>
45  <body>
46      <h1 oncllck="stop_watch();">获取当前位置信息</h1>
47      <div class="line"></div>
48      <h2>你的当前位置: </h2>
49      <h4 id="Longitude">经度: </h4>
50      <h4 id="Latitude">纬度: </h4>
51      <h4 id="Altitude">海拔高度: </h4>
52      <h4 id="Accuracy">精度: </h4>
```

```
53          <h4 id="Altitude_Accuracy">海拔准确度：</h4>
54          <h4 id="Heading">航向：</h4>
55          <h4 id="Speed">速度：</h4>
56          <h4 id="Timestamp">时间：</h4>
57      </body>
58      </html>
```

通过范例不难看出，该方法与在上一节中介绍过的 getCurrentPosition()方法在用法上非常类似，只是增加了一个参数 option。那么参数 option 又是干什么的呢？

从 Cordova 提供的文档中可以了解到，这里的 option 实际上是一个 geolocationOptions 类型的可选参数，用来定义对程序获取位置信息的一些设置。它的具体属性以及说明参见表 9-2。

表 9-2　option 对象中的属性

名称	类型	说明
frequency	整数型	用来定义获取设备位置信息的频率，单位是毫秒
enableHighAccuracy	布尔类型	提供一个表明应用程序希望获得最佳可能结果的提示，可以理解为是否希望使用高精确度的定位
timeout	整数型	两次获取设备位置信息所允许的最大时间间隔，单位为毫秒
maximumAge	整数型	应用程序将接受一个缓存的位置信息，该缓存的位置信息的年龄不大于此参数设定值，单位是毫秒。



实际上 maximumAge 属性的出现是由于最初对 frequency 的定义不符合 W3C 规范，因此设计了这个参数计划在将来用它来取代 frequency。

如果说 watchPosition()的用法就只有这些，也许已经足够了；但是在某些应用上却可能会有一些瑕疵，比如当用户想要停止获取设备位置时该怎么办呢？总不可能让用户去“后台”强行关闭掉程序吧。

事实上还有一个方法是笔者没有介绍到的，那就是在范例第 33 行用到的 clearWatch()方法。该方法只需要一个参数，就是当前所监视位置信息所使用的 id，那么这个 id 又是从哪里来的呢？再回过头来看范例第 14 行。在 watchPosition()方法开始执行时会返回一个 id 数值，而在本范例中将这个数值保存在了变量 watchID 中，因此当想要停止对设备位置的监控时只需要执行：

```
navigator.geolocation.clearWatch(watchID);
```

说到这里，笔者突然想到了一种非常不错的表白方法，读者可以去尝试一下，当然还是利用本节中介绍的 watchPosition()方法来实现。既然能够利用 watchPosition()方法实现对设备位置信息的获取，那么如果把具有同样类型的应用装在女神的手机里会怎样呢？

当然这不是教唆读者利用这种方法去监视女神的动态，然后半路拦截；而是有更加浪漫

的方法。既然能够获取女神的位置了，是不是可以将这些坐标在屏幕上显示出来呢？那么如果拿着女神的手机在田野里绕着一个心形的图案跑一圈屏幕上是不是就会显示出一个心形呢？

好了，话就说这么多，能不能成功就看各位读者的动手能力了。

9.4 设备方向的获取

本章前面的内容介绍了怎样获取设备的位置信息，不知道读者在学完了这些内容之后，有没有考虑过这些 API 能够实现什么样的功能呢？反正笔者当时思考了很久，结果近乎荒诞地想到了指南针。这是一个非常愚蠢的想法，因为从坐标信息到方向信息完全是两个不搭边的功能。但是这也说明了一个问题，指南针作为移动设备在地理信息方面的应用实在是太深入人心了，如图 9-3 所示的就是一款指南针应用的界面。



图9-3　一款指南针应用的界面

有过开发应用经验的读者可能都知道，想要开发一个具有动画效果（如指南针的转动）的应用是很难的，但是如果用 HTML 5 就能够比较轻松地实现这样的动画效果。那么是不是可以用 Cordova 来实现一款指南针呢？答案是肯定的。

如果要实现指南针的功能，就必须要有能够获取设备方向的 API，这正是在本节所要介绍的内容。

首先在命令行执行命令添加插件:cordova plugin add cordova-plugin-device-orientation。

【范例 9-3】利用 getCurrentHeading()方法获取设备的当前朝向。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta charset="utf-8">
05  <title>利用 getCurrentHeading()方法获取设备的当前朝向</title>
06  <script src="js/cordova.js" type="text/javascript"></script>
07  <script>
```

```
08      // 设置当设备加载完毕后执行的触发函数 onDeviceReady
09      document.addEventListener("deviceready", onDeviceReady, false);
10      function onDeviceReady() {
11          // 设备加载完毕后执行 getCurrentHeading()方法
12          navigator.compass.getCurrentHeading(onSuccess, onError);
13      }
14      // 当获取设备方向后执行该函数
15      function onSuccess(heading) {
16          // 弹出消息框显示当前设备方向
17          alert("当前设备方向与正北方相差" + heading.magneticHeading + "°");
18      }
19      // 当获取设备方向失败时执行该函数
20      function onError(compassError) {
21          alert("获取设备方向失败");
22      }
23  </script>
24  </head>
25  <body>
26      <h1>显示当前的方向</h1>
27  </body>
28  </html>
```

运行结果如图 9-4 所示，得出的结果是笔者的手机所朝的方向。



图 9-4　手机上获取了当前设备方向



读者在使用 Cordova 的指南针功能时一定要在真机上进行测试，否则会获取失败。

getCurrentHeading()方法的使用与其他 Cordova 中的 API 非常类似，依然是在回调函数 onSuccess 中对获取到的结果进行处理。该方法传递来的结果是一个 heading 对象，它包含了表 9-3 所示的 4 个属性。

表 9-3 heading 对象中的属性

名称	类型	说明
magneticHeading	数字	用来表示设备在某一时刻相对于正北方向偏移的角度，它的值从 0~359.99 不等
trueHeading	数字	与 magneticHeading 相同，但是当当前的方向不确定时，比如你现在正在北极，那么该属性的值将是负的
headingAccuracy	数字	测量的方向可能存在的误差
timestamp	整数	用来记录获取设备方向的时间，单位是毫秒

9.5 监视设备方向的两种方法

上一节介绍了使用 Cordova 获取设备方向的方法，细心的读者可能已经发现笔者在本节要介绍的两种传感器了。但是为什么要把它们合在一起介绍呢？除了都可以用来获取与地理位置有关的信息之外，也许还会有其他的原因。

那就是这两种传感器的用法实在是太相近了，在获取设备的位置信息时，可以使用 getCurrentPosition()和 watchPosition()，而在获取设备的方向时除了可以使用上一节中介绍的 getCurrentHeading()之外，还可以使用另一个方法 watchHeading()来对设备的方向进行监视。

【范例 9-4】使用 watchHeading()方法监视设备方向。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta charset="utf-8">
05  <title>使用 watchHeading()方法监视设备方向</title>
06  <script src="js/cordova.js" type="text/javascript"></script>
07  <script>
08      var watchID = null;
09      // 设置当设备加载完毕后执行的触发函数 onDeviceReady
10      document.addEventListener("deviceready", onDeviceReady, false);
11      function onDeviceReady() {
12      // 设备加载完毕后执行 getCurrentHeading()方法,此处的 option 与第2节中介绍的相同
13          var options = { frequency: 500 };
14          watchID = navigator.compass.watchHeading(onSuccess, onError,
options);
15     }
16      // 当获取设备方向后执行该函数
17      function onSuccess(heading) {
```

```
18            // 弹出消息框显示当前设备方向
19            document.getElementById("heading").innerHTML = "当前设备方向是：" +
heading.magneticHeading;
20            document.getElementById("time").innerHTML = heading.timestamp;
21        }
22        // 当获取设备方向失败时执行该函数
23        function onError(compassError) {
24            alert("获取设备方向失败");
25        }
26        // 调用该函数停止对设备方向的监视
27        function stop_watch() {
28            if(watchID) {
29                navigator.compass.clearWatch(watchID);
30            }
31        }
32    </script>
33    <style>
34    .line
35    {
36        width:100%;
37        height:1px;
38        border:1px solid #000;
39    }
40    </style>
41    </head>
42    <body>
43        <h1 onclick="stop_watch();">显示当前的方向</h1>
44        <div class="line"></div>
45        <h2 id="heading"></h2>
46        <h2 id="time"></h2>
47    </body>
48    </html>
```

运行结果如图 9-5 所示，如果点击屏幕顶部的“显示当前方向”字样，则会停止对设备方向的监视。



图 9-5　监视设备方向所得的结果

通过对范例的阅读可以发现，watchHeading()的使用与 9.3 节介绍的 watchPosition()方法是一样的，因此这里就不再对它的用法做过多的重复介绍了。这里要说的是除了 watchHeading()，在 Cordova 中还有一种方法能够实现对方向的监控，那就是 watchHeadingFilter()方法，它的作用是当设备方向发生变化时执行相对应的 onSuccess 函数。

【范例 9-5】使用 watchHeadingFilter()方法监视设备方向。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta charset="utf-8">
05  <title>使用 watchHeadingFilter()方法监视设备方向</title>
06  <script src="js/cordova.js" type="text/javascript"></script>
07  <script>
08      var watchID = null;
09      // 设置当设备加载完毕后执行的触发函数 onDeviceReady
10      document.addEventListener("deviceready", onDeviceReady, false);
11      function onDeviceReady() {
12          // 当设备方向改变10°以上时设备获取通知
13          var options = { filter: 10 }
14          watchID = navigator.compass.watchHeadingFilter(onSuccess, onError,
options);
15      }
16      // 当获取设备方向后执行该函数
17      function onSuccess(heading) {
18          // 弹出消息框显示当前设备方向
19          document.getElementById("heading").innerHTML = "当前设备方向是: " +
heading.magneticHeading;
20          document.getElementById("time").innerHTML = heading.timestamp;
```

```
21          }
22          // 当获取设备方向失败时执行该函数
23          function onError(compassError) {
24              alert("获取设备方向失败");
25          }
26          // 调用该函数停止对设备方向的监视
27          function stop_watch() {
28              if(watchID) {
29                  navigator.compass.clearWatchFilter(watchID)
30              }
31          }
32      </script>
33      <style>
34      .line
35      {
36          width:100%;
37          height:1px;
38          border:1px solid #000;
39      }
40      </style>
41      </head>
42      <body>
43          <h1 onclick="stop_watch();">显示当前的方向</h1>
44          <div class="line"></div>
45          <h2 id="heading"></h2>
46          <h2 id="time"></h2>
47      </body>
48      </html>
```

运行结果如图 9-6 所示，读者可以通过静止手机或是将手机转动一定的角度来观察屏幕上数据的变化。



图 9-6　使用 watchHeadingFilter()方法监视设备方向

从范例中可以看出，该 watchHeadingFilter()方法在使用上与范例 9-4 中的 watchHeading()

方法非常类似，但是在第 13 行中可以看到，该方法的 option 参数发生了变化，因为在此处所使用的 option 变成了一个 compassOptions 类型的对象，开发者可以通过它来设置当设备变化了多少角度时程序才会有所反应。

接下来对这两种方法进行比较，不过既然同时存在两种方法，而没有废除掉其中的一种，那就说明两种方法都存在可取的地方。watchHeading()方法用来按照一定的时间周期对设备的方向进行监视，这就会导致一个问题，如果这个频率设得太高，比如每隔 1ms 就获取一次方向数据，即使是八核的 CPU 也会有些吃力；而如果频率设得太低，则获取的数据又不够精确。

使用 watchHeadingFilter()方法就不需要考虑这样的问题，但是却会遇到另一种麻烦，比如将 option 参数中的值设的过大，如范例中的 10，就会造成设备不够灵敏。但是一般来说将这个值设为 1 就足够使用了。但是假设此时遇到了一个爱动的用户，它尝试让手机在指尖旋转，那么就会发生可怕的事情，虽然手机不会爆炸，但至少这个程序很难再保持正常运行了。

9.6 小结

本章主要学习了使用 Cordova 如何获取设备的位置信息以及方向的信息，再加上在第 8 章介绍过的加速度传感器，相信读者已经能够想出许多令人激动的玩法了。通过本章的学习，读者还应当能够体会到 Cordova 所提供的 API 在用法上的一些共同思路，理解了这种思路之后，Cordova 的学习之路就会容易许多。

第 10 章

Cordova对音频的控制

除了设备传感器，对多媒体文件的处理也是智能手机一个非常重要的功能。像酷我音乐盒、酷狗音乐等音乐播放器都是每一台智能手机上必装的应用。因此在开发智能手机应用时，对音频文件的处理是每一个开发者都必须掌握的。在 Cordova 中，Media 对象提供了对音频文件的播放和录制的功能。本章还将实现一个简单的录音机，能够实现对音频文件的录制、播放和暂停功能。

本章的主要知识点有：

- 利用 Cordova 播放和暂停音频文件的方法。
- 获取音频文件资源的方法。
- 利用 Cordova 录制音频文件的方法，以及播放所录制的音频文件的方法。

10.1 利用 Cordova 播放音频的方法

本节将先以一个例子来播放一段来自于网络的音乐，如范例 10-1 所示。

首先在命令行执行命令添加插件：cordova plugin add cordova-plugin-media。

【范例 10-1】利用 Cordova 播放一段音乐。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>利用 Cordova 播放音乐</title>
06  <script type="text/javascript" charset="utf-8" src="js/cordova.js">
</script>
07  <script type="text/javascript" charset="utf-8">
08      // 等待加载 Cordova
09      document.addEventListener("deviceready", onDeviceReady, false);
10      // Cordova 加载完毕
11      function onDeviceReady() {
```

```
12              // 设备加载完毕
13              alert("设备加载完毕");
14          }
15          // 音频播放器
16          var my_media = null;
17
18          // 播放音频
19          function playAudio() {
20              // 从目标文件创建 Media 对象
21              src = "http://audio.ibeat.org/content/p1rj1s/p1rj1s_-
_rockGuitar.mp3";
22              my_media = new Media(src, onSuccess, onError);
23              // 播放音频
24              my_media.play();
25          }
26          // 创建 Media 对象成功后调用的回调函数
27          function onSuccess() {
28              alert("playAudio():Audio Success");
29          }
30          // 创建 Media 对象出错后调用的回调函数
31          function onError(error) {
32              alert('code: '    + error.code    + '\n' +
33                  'message: ' + error.message + '\n');
34          }
35      </script>
36      </head>
37      <body>
38          <h1 onclick="playAudio();">利用 Cordova 播放音乐</h1>
39      </body>
40      </html>
```

程序运行之后，点击屏幕上的“利用 Cordova 播放音乐”字样，就可以听到有音乐从手机扬声器中发出，并将弹出对话框，内容为 playAudio():Audio Success，用以提示用户 Media 对象创建成功。接下来将结合范例代码讲解在 Cordova 中利用 Media.play()方法播放音乐的方法。

在 Cordova 中，Media 对象为开发者提供了控制设备音频的能力，其中 paly()方法可以用来播放指定的音频文件。在使用 play()方法时，首先要指定音频文件的来源，如范例第 21 行所示。由于 play()方法同时可以支持本地和网络的音频文件，为了便于理解可以使用网络上音频文件的 url 地址。

第 22 行创建了一个新的 Media 对象，与 Cordova 中的大多数其他方法一样，在创建 Media 对象时需要定义回调函数。



在最初的时候，Cordova 处理音频的 API 并没有遵循 W3C 规则，仅仅是为开发者提供了一个方便调用的接口，但是随着版本的更新和 Cordova 的不断完善，目前这一现象已经得到了很好的改善。

创建一个 Media 对象的参数有 4 个，以下是官方对该对象的声明：

```
var media = new Media(src, mediaSuccess, [mediaError], [mediaStatus]);
```

在该声明中有 4 个参数，其中音频文件的源地址和 Media 对象创建成功的回调函数 mediaSuccess()是必选的，而其他两个参数是可选的。可选参数中 mediaError()函数用来对 Media 对象创建失败的情况进行操作，mediaStatus()是在音频播放状态发生变化时所调用的函数。

接下来就可以调用 Media 的 play()方法对音频进行播放了，如范例第 24 行所示。

```
24          my_media.play();
```



在虚拟机上进行测试时，如果获取来自网络上的资源速度较慢，可以在 DDMS 中将 Speed（网速）设置为 Fast（快速），如图 10-1 所示。

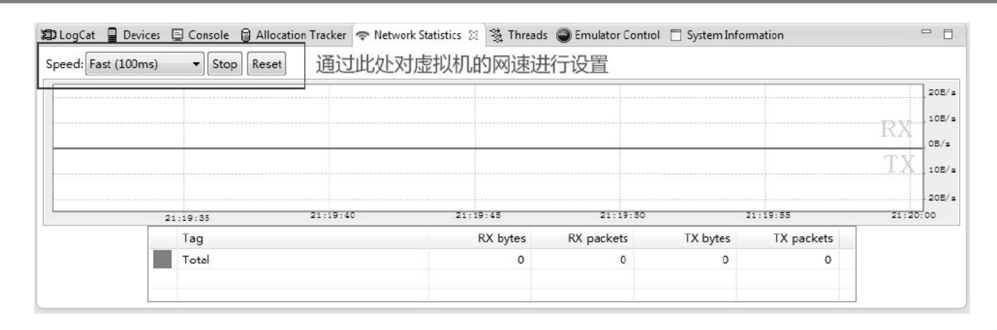


图 10-1　在 DDMS 中模拟设备的网速

10.2 利用 pause()方法暂停播放音乐

上一节学习了利用 Media 对象的 paly()方法来播放音乐，但是在测试范例 10-1 时遇到了一点小问题，就是一旦音乐开始播放，那声音就像春晚的小彩旗一样根本停不下来。试想，如果做了这样一款音乐播放器，会有用户喜欢吗？所以，除了播放之外，暂停已经播放了的音乐也是在实际开发中非常重要的一个部分。

【范例 10-2】利用 pause()方法暂停播放音乐。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>利用 pause()方法暂停音乐</title>
06  <script type="text/javascript" charset="utf-8" src="js/cordova.js">
</script>
07  <script type="text/javascript" charset="utf-8">
08      // 等待加载 Cordova
09      document.addEventListener("deviceready", onDeviceReady, false);
10      // Cordova 加载完毕
11      function onDeviceReady() {
12          // 设备加载完毕
13          alert("设备加载完毕");
14      }
15      // 音频播放器
16      var my_media = null;
17      var mediaTimer = null;
18      // 播放音频
19      function playAudio() {
20          // 从目标文件创建 Media 对象
21          src = "http://audio.ibeat.org/content/p1rj1s/p1rj1s_-
_rockGuitar.mp3";
22          my_media = new Media(src, onSuccess);
23          // 播放音频
24          my_media.play();
25      }
26      // 暂停音频
27      function pauseAudio() {
28          if(my_media) {
29              my_media.pause();
30          }
31      }
32      // 创建 Media 对象成功后调用的回调函数
33      function onSuccess() {
34          alert("playAudio():Audio Success");
35      }
```

```
36 </script>
37 </head>
38 <body>
39     <h1 onclick="playAudio();">利用 Cordova 播放音乐</h1>
40     <h1 onclick="pauseAudio();">暂停音乐播放</h1>
41 </body>
42 </html>
```

程序运行之后，点击屏幕上的“利用 Cordova 播放音乐”字样，就可以听到有音乐从手机扬声器中发出，再点击“暂停音乐播放”字样，则音乐会被暂停。其原因就在于点击“暂停音乐播放”后会调用自定义函数 pauseAudio()，而在 pauseAudio()函数中则会调用 Media 对象的 pause()方法（范例第 29 行所示）。

看第 28 行会发现，在 pause()方法外还有一个 if 判断，它的作用是在暂停音乐之前先判断 Media 对象是否确实存在，在实际开发中如果要改变某一对象的状态，先判断它的存在性是非常重要的。

10.3 利用 stop()方法停止播放音频文件

上一节学习了利用 pause()暂停播放音频文件的方法，本节将继续学习一种与 pause()非常类似却又有不少区别的方法：stop()。

【范例 10-3】利用 stop()方法停止音乐播放。

```
01 <!DOCTYPE html>
02 <html>
03 <head>
04 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05 <title>利用 stop()方法停止音乐播放</title>
06 <script type="text/javascript" charset="utf-8" src="js/cordova.js">
   </script>
07 <script type="text/javascript" charset="utf-8">
08     // 等待加载 Cordova
09     document.addEventListener("deviceready", onDeviceReady, false);
10     // Cordova 加载完毕
11     function onDeviceReady() {
12         // 设备加载完毕
13         alert("设备加载完毕");
14     }
15     // 音频播放器
```

```
16      var my_media = null;
17      var mediaTimer = null;
18      // 播放音频
19      function playAudio() {
20          // 从目标文件创建 Media 对象(也可以使用本地文件)
21          src = 'img/K. Williams - 菊次郎的夏天.mp3';
22          if(my_media == null){
23          my_media = new Media(src, onSuccess);
24          }
25          // 播放音频
26          my_media.play();
27      }
28      // 创建 Media 对象成功后调用的回调函数
29      function onSuccess() {
30          alert("playAudio():Audio Success");
31      }
32      // 暂停音频
33      function pauseAudio() {
34          if(my_media) {
35              my_media.pause();
36          }
37      }
38      // 停止音频
39      function stopAudio() {
40          if(my_media) {
41              my_media.stop();
42          }
43      }
44  </script>
45  </head>
46  <body>
47      <h1 onclick="playAudio();">利用 Cordova 播放音乐</h1>
48      <h1 onclick="pauseAudio();">暂停音乐播放</h1>
49      <h1 onclick="stopAudio();">停止音乐播放</h1>
50  </body>
51  </html>
```

运行之后可以通过点击屏幕上的“停止音乐播放”字样来停止正在播放中的音乐。与pause()方法不同的是，使用 pause()方法暂停的音乐可以通过 play()方法继续播放，而使用

stop()方法停止播放的音乐想要再进行播放，将会从头开始。

可以再在页面中加入 pause()方法来比较两方法的不同，如图 10-2 所示。

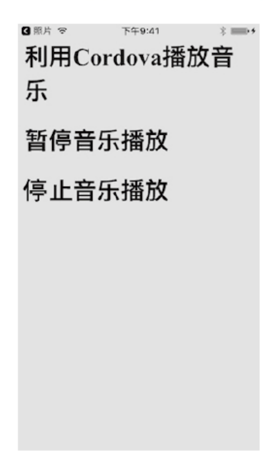


图 10-2　对比 pause()和 stop()方法的不同

10.4 获取音频文件的更多信息

在音乐播放器的界面上通常都会有一条“线”，用来表示音乐播放的总时长和进度（如图 10-3 所示）。如果想要实现这样的功能，就需要获得音频文件的总长度以及当前音频文件播放的进度。Cordova 也提供了这样的方法。

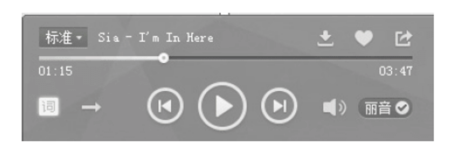


图 10-3　酷狗音乐中的进度条

media.getDuration()方法的作用是返回一个音频文件的总时长，media.getCurrentPosition()的作用是返回当前音频文件所播放到的位置，范例 10-4 将使用它们来获取一首歌的总时长以及播放进度。

【范例 10-4】获取音乐的总时长以及播放进度。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>利用 Cordova 播放音乐</title>
06  <script type="text/javascript" charset="utf-8" src="js/cordova.js">
```

```
    </script>
07  <script type="text/javascript" charset="utf-8">
08      // 等待加载 Cordova
09      document.addEventListener("deviceready", onDeviceReady, false);
10      // Cordova 加载完毕
11      function onDeviceReady() {
12          // 设备加载完毕
13          alert("设备加载完毕");
14      }
15      // 音频播放器
16      var my_media = null;
17      var mediaTimer = null;
18      // 存放音乐时间总长
19      var dur = null;
20      // 当前播放时间
21      var midtimer = null;
22      // 播放音频
23      function playAudio() {
24          // 从目标文件创建 Media 对象
25          src = "http://audio.ibeat.org/content/p1rj1s/p1rj1s_-_rockGuitar.mp3";
26          my_media = new Media(src, onSuccess);
27          // 播放音频
28          my_media.play();
29          // 每50ms 获取一次播放位置
30          dur = my_media.getDuration();
31          if( midtimer == null) {
32              midtimer = setInterval( function() {
33                  my_media.getCurrentPosition(
34                  // 获取进度成功
35                  function(position) {
36                      if( position > -1 ) {
37                          setAudioPosition((position) + "/" + (dur) );
38                      }
39                  }
40                  ),
41                  // 获取进度失败
42                  function(e) {
43                      alert("获取播放进度失败");
44                  }
```

```
45                }
46                ,50);
47            }
48        }
49        // 创建 Media 对象成功后调用的回调函数
50        function onSuccess() {
51            alert("playAudio():Audio Success");
52        }
53        // 停止音频
54        function stopAudio() {
55            if(my_media) {
56                my_media.stop();
57            }
58        }
59        function setAudioPosition(position) {
60              document.getElementById('audio_position').innerHTML = position;
61        }
62    </script>
63    </head>
64    <body>
65        <h1 id="audio_position"></h1>
66        <h1 onclick="playAudio();">利用 Cordova 播放音乐</h1>
67        <h1 onclick="stopAudio();">停止音乐播放</h1>
68    </body>
69    </html>
```

运行结果如图 10-4 所示。

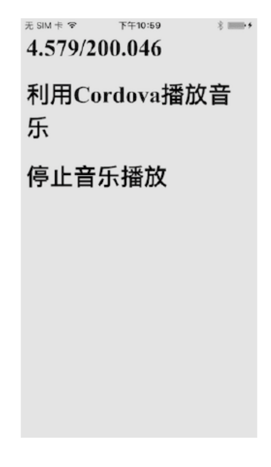


图 10-4　获取音频文件的播放位置以及总长度

getDuration()和 getCurrentPosition()的返回值都是一个单位为秒的小数，分别用来表示音乐的总长度以及当前播放进度。若想要获取音频文件的总长度，可直接使用 getDuration()方法，如范例第 30 行所示。理论上来说，getCurrentPosition()也是如此使用，但是由于它的作用是返回音乐当前播放位置，这就决定了它用来代表一个动态的过程。

因为音频文件是在不断播放的，因此不可能保证当前屏幕上显示的时间就是当前的播放进度，为了尽量精确就需要使用 JavaScript 中的 setInterval()方法来配合，如范例第 32 行所示。

当使用了 getCurrentPosition()后，所获得的数据会被保存在对象 position 中，此时就可以利用获取播放进度成功的回调函数来处理获得的数据。

10.5 播放指定位置的音乐

上一节介绍了利用 Cordova 获取音频文件总长度以及当前播放位置的方法，通过这两个方法再加上 CSS 和 JavaScript 的配合就已经能够完成一个不错的进度条了。相比目前比较成熟的应用中的进度条，比如酷狗音乐，还缺少了利用进度条来调整播放进度的功能。但是没有关系，本节将介绍一个方法来弥补这个不足。

【范例 10-5】利用 seekTo()方法从指定位置播放音乐。

```
01  <!DOCTYPE HTML>
02  <html>
03  <head>
04      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05      <title>利用 seekTo()方法从指定位置播放音乐</title>
06      <script type="text/javascript" charset="utf-8" src="js/cordova.js">
        </script>
07      <script type="text/javascript" charset="utf-8">
08      // 设置设备加载的触发器
09      document.addEventListener("deviceready", onDeviceReady, false);
10      // 设备加载完毕
11      function onDeviceReady() {
12          // 播放音乐
13          playAudio("202.118.86.5/test.mp3");
14      }
15      // 设置用来记录播放进度的变量
16      var my_media = null;
17      var mediaTimer = null;
18      // 播放音乐
19      function playAudio(src) {
```

```
20        // 创建新的 Media 对象
21        my_media = new Media(src, onSuccess, onError);
22        // 播放
23        my_media.play();
24        // 使用计数器更新播放进度
25        mediaTimer = setInterval(function() {
26        // 获取播放进度
27        my_media.getCurrentPosition(
28        // 获取播放进度成功
29            function(position) {
30                if (position > -1) {
31                    setAudioPosition(position + " sec");
32                }
33            },
34            // 获取播放进度失败
35            function(e) {
36                alert("Error getting pos=" + e);
37            }
38        );
39        }, 1000);
40        // 跳至10s 处开始播放
41        setTimeout(function() {
42            my_media.seekTo(10000);
43        }, 5000);
44        }
45        // 停止播放音乐
46        function stopAudio() {
47            if (my_media) {
48                my_media.stop();
49            }
50            clearInterval(mediaTimer);
51            mediaTimer = null;
52        }
53        function onSuccess() {
54            console.log("playAudio():Audio Success");
55        }
56        function onError(error) {
57            alert('code: '    + error.code    + '\n' +
58                    'message: ' + error.message + '\n');
```

```
59        }
60        // 显示当前播放进度
61        function setAudioPosition(position) {
62            document.getElementById('audio_position').innerHTML = position;
63        }
64    </script>
65    </head>
66    <body>
67        <h1 href="#" class="btn large"
onclick="playAudio('202.118.86.5/test.mp3'); ">播放</h1>
68        <h1 href="#" class="btn large" onclick="stopAudio();">停止</h1>
69        <h1 id="audio_position"></h1>
70    </body>
71    </html>
```

运行之后音乐会自动播放，当播放到 5s 和 10s 处时，音乐会有一个短暂的停顿发生，图 10-5 为第 10s 音乐刚从停顿中恢复时的截图。



图 10-5　音乐播放至 10s 时的截图



具体的效果还请读者实际运行程序来感受，但是笔者要说的是：由于运行效率等方面的原因，在 Cordova 中的计时并不十分精确，甚至有着较大的误差。因此在使用此类功能时一定要做好测试，避免可能存在的误差。

10.6 使用 Cordova 录制声音

Media 对象除了可以播放音频之外，还有一个重要的功能就是可以实现对音频的录制，这一功能是通过 Media 对象的 startRecord()方法和 stopRecord()方法实现的。

在苹果公司看来录音功能会涉及隐私敏感数据，所以如果你想要调用用户手机的录音功能的话要先获取用户的同意才可以，方法是在工程的 info.plist 文件里面添加相应的字段 NSMicrophoneUsageDescription，如图 10-6 所示。

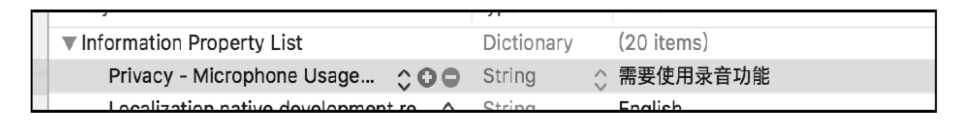


图 10-6　info.plist 文件配置信息

该字段后对应的值是 String 类型，可以形象说明用户为什么要使用该功能。

【范例 10-6】利用 Cordova 实现录音功能。

```
01  <!DOCTYPE HTML>
02  <html>
03  <head>
04      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05      <title>利用 Cordova 实现录音功能</title>
06      <script type="text/javascript" charset="utf-8" src="js/cordova.js">
</script>
07      <script type="text/javascript" charset="utf-8">
08      // 等待加载 Cordova
09      document.addEventListener("deviceready", onDeviceReady, false);
10      var mediaRec = null
11      function recordAudio() {
12          var src = "myrecording.wav";//  ios 录音必须使用 wav 模式
13           mediaRec = new Media(src, onSuccess, onError);
14          // 开始录制音频
15          mediaRec.startRecord();
16          // 10s 后停止录制
17          var recTime = 0;
18          var recInterval = setInterval(function() {
19              recTime = recTime + 1;
20              setAudioPosition(recTime + " sec");
21              if (recTime >= 10) {
22                  clearInterval(recInterval);
23                  mediaRec.stopRecord();
24              }
25          }, 1000);
26      }
27      // Cordova 加载完毕
28      function onDeviceReady() {
29          alert("加载完成");
30      }
31      // 创建 Media 对象成功后调用的回调函数
```

```
32        function onSuccess() {
33            console.log("recordAudio():Audio Success");
34        }
35        // 创建 Media 对象出错后调用的回调函数
36        function onError(error) {
37            alert('code: '    + error.code    + '\n' +
38                  'message: ' + error.message + '\n');
39        }
40        // 设置音频播放位置
41        function setAudioPosition(position) {
42            document.getElementById('audio_position').innerHTML = position;
43        }
44        // 播放音频
45        function playAudio() {
46            mediaRec.play();
47        }
48    </script>
49    </head>
50    <body>
51        <h1 onclick = "recordAudio()">开始录音</h1>
52        <h1 onclick = "playAudio()">播放</h1>
53        <h1 id = "audio_position"></h1>
54    </body>
55    </html>
```

运行之后点击屏幕上的“开始录音”字样即可开始录音，底部开始进行计数，当计数到10时录音结束，可以点击“播放”字样播放刚刚录下的声音，如图10-7所示。

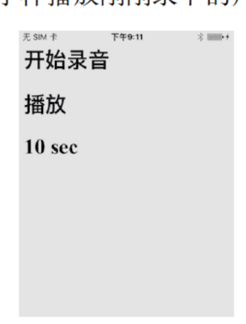


图10-7　录音和播放界面

与播放音频一样，在录制音频时也需要先定义一个Media对象，如第12、13行所示，不同的是，第一个参数必须要定义一个文件用来存放录制的音频，本例中为myrecording.wav。之后就可以使用startRecord()方法对声音进行录制了，如第15行所示。

接下来又使用了一个 setInterval()方法，它的作用是控制录音时间的长短，但是由于在 Cordova 中计数器实现的时间并不十分精确（纯 JavaScript 中该函数所计算出的时间都不是非常精确），因此在实际开发中建议还是手动来结束录音。



如果要定时录制一段较长的音频，可以通过不断获取系统时间的方法来达到较精确定时的目的。

10.7 释放音频资源

在 C 语言中，如果要调用一段内存可以使用函数 malloc()来达到目的，而用过之后则可以使用 free()函数来释放这一段内存，比如：

```
int *p;
p = (int)malloc(sizeof(int));
// 对 p 进行各种计算
free(p);
```

而在 Cordova 中也有着类似的操作。由于无论是设备还是受 JavaScript 效率的限制，系统所能使用的资源都是有限的，因此在资源不用时要及时进行回收。Cordova 的 Media 对象并没有专门用来释放资源的方法，因此当需要替换原有资源时会造成浪费，这时可以使用 Media 对象的 release()方法。



尤其是在安卓系统中，由于系统的多媒体核心有限，如果连续新建多个 Media 对象，就可能造成音乐无法播放甚至是系统错误。

10.8 实战：制作一个简单的“录音机”软件

本章已经学习了利用 Cordova 进行音频处理的一些方法，尤其是 10.4 节中还用一个比较复杂的例子演示了获取和显示播放进度的方法。但是本章给出的例子都过于粗糙，整个界面上就只有几行简单的文字，给人一种粗制滥造的印象。那么本节将进行一个比较复杂的实战，来实现录音机 APP。

10.8.1　需求分析

虽然本节的例子比较复杂，但是由于篇幅限制仍然不会加入新的知识，主要是实现录音和播放功能，并将这一功能在一套完整的界面中实现。为了能有更好的视觉效果，本次使用的例子将为播放功能加入进度条的视觉效果，而录音功能也要加入一个简单的计时功能。

本例所使用的界面结构如图 10-8 所示。

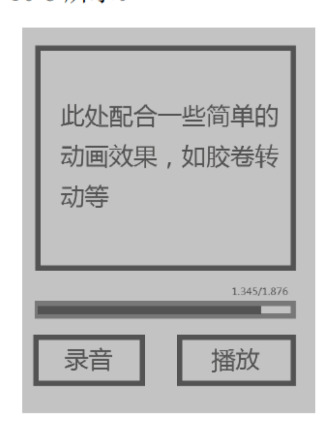


图 10-8　录音机的界面设计

由于界面比较简单且没有专职的美工，为了达到“遮丑”的效果在配色上就需要用一些有强对比度的颜色，比如全黑背景加上亮度较高的绿色或者蓝色（黑客帝国数字雨的那种配色）。虽然一直有开发者抱怨这样的方案会显得非常山寨、非常低级，但不得不说在缺少全职美工又没有 UI 素材可用时，这就是最好的方案了。



由于使用太多素材不便于读者学习，因此在范例中笔者会尽量不使用图片等素材，以确保将最基础的内容展现给读者。在实际开发中，即使没有美工，也有许多 UI 插件（比如 jQuery Mobile）可以供个人开发者使用，使得没有美工基础的开发者也能做出不错的界面。

程序运行后，可以点击“录音”按钮对声音进行录制。此时，该按钮上的“录音”字样变为“停止”，界面上方将会播放动画代表录音正在进行中，进度条上方的时间标签显示录音时间。当录音完成后，点击“停止”录音结束。

“播放”按钮也是同理，而且随着播放的进行，屏幕中央的进度条也会随之变化来响应播放的进度。

10.8.2　界面实现

准备一张图片，长宽均为 160px，背景透明，如图 10-8 所示作为顶部“播放”动画效果的原形素材，命名为 rec.png。再新建文件命名为 index.html，具体内容如范例 10-7 所示。

【范例 10-7】界面实现之 index.html 部分。

```
01  <!DOCTYPE HTML>
02  <html>
03  <head>
04      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05      <title>录音界面</title>
```

```
06      <script type="text/javascript" charset="utf-8" src="js/cordova.js">
</script>
07      <!--引入样式文件-->
08      <link rel="stylesheet" type="text/css" href="style.css">
09  <script type="text/javascript" charset="utf-8">
10      // 此处加入 JavaScript 代码
11  </script>
12  </head>
13  <body>
14      <div id="rec">
15          <!--顶部-->
16          <div id="top">
17              <div id="top_rec">
18                  <!--顶部动画图片-->
19                  <div id="pic"></div>
20              </div>
21          </div>
22          <!--中部-->
23          <div id="mid">
24              <div id="mid_rec">
25                  <!--显示时间-->
26                  <div id="mid_rec_num">1.222/2.222 S </div>
27                  <div id="mid_rec_bar">
28                      <div id="bar"></div>
29                  </div>
30              </div>
31          </div>
32          <!--底部-->
33          <div id="bottom">
34              <!--底部两按钮-->
35              <div id="bottom_rec">
36                  <div id="button_left">录音</div>
37                  <div id="button_right">播放</div>
38              </div>
39          </div>
40      </div>
41  </body>
42  </html>
```

再新建一个文件 style.css，内容如范例 10-8 所示。

【范例 10-8】界面实现之 style.css 部分。

```
/**清空默认边距**/
* { margin:0px; padding:0px; }
/**将#rec 至于中间位置，保证兼容性**/
body  { margin:auto; }
/**根据测试机型设置固定大小，暂时不考虑其他机型的适配**/
#rec
{
    margin:auto;
    width:320px;
    height:510px;
    background:#000;
}
/**顶部**/
#top
{
    width:320px;
    height:270px;
    float:left;
}
/**中间**/
#mid
{
    width:320px;
    height:80px;
    float:left;
}
/**底部**/
#bottom
{
    width:320px;
    height:100px;
    float:left;
}
/**用来显示顶部的边框**/
#top_rec
{
```

```
    width:200px;
    height:200px;
    border:3px solid #00ff00;
    margin:50px 57px 0 57px;
}
/**显示顶部动画图片**/
#pic
{
    width:160px;
    height:160px;
    margin:20px;
    background:url(rec.png);
}
/**显示中间边框**/
#mid_rec
{
    width:300px;
    height:40px;
    border:3px solid #00ff00;
    margin:45px 7px 0 7px;
}
/**用于显示进度信息**/
#mid_rec_num
{
    width:300px;
    height:24px;
    line-height:24px;
    color:#00ff00;
    text-align:right;
    font-size:18px;
    float:left;
}
/**进度条的绿色背景**/
#mid_rec_bar
{
    width:300px;
    height:16px;
    background:#ffff00;
    float:left;
```

```
}
/**黄色进度条**/
#bar
{
    width:20px;
    height:16px;
    background:#00ff00;
    float:left;
}
/**底部外框**/
#bottom_rec
{
    width:300px;
    height:50px;
    margin:30px 10px 0 10px;
}
/**左侧的按钮**/
#button_left
{
    width:134px;
    height:40px;
    float:left;
    border:3px solid #00ff00;
    border-radius:15px;
    -webkit-border-radius:15px;
    font-size:26px;
    color:#00ff00;
    text-align:center;
    line-height:40px;
}
/**右侧的按钮**/
#button_right
{
    width:134px;
    height:40px;
    float:left;
    border:3px solid #00ff00;
    margin-left:20px;
    border-radius:15px;
```

```
    -webkit-border-radius:15px;
    font-size:26px;
    color:#00ff00;
    text-align:center;
    line-height:40px;
}
```

运行之后的效果如图 10-9 所示。



图 10-9　实现后的界面效果

在实际使用时，顶部胶带轮盘的图片是可以转动的，从而形成一种正在进行播放或者是录音的状态效果，底部的进度条以及按钮上的字样也是根据当前的应用状态自行改变的。

在本项目中，大致分为以下几个状态：

- 未录音：应用处于运行状态，但是未进行“录音”操作，两按钮内容分别为“录音”和“播放”，由于未进行录音操作，“播放”功能不可用。
- 录音中：应用处于运行状态，正在进行录音，两按钮内容分别为“停止”和“播放”，顶部胶带轮盘转动，“播放”功能不可用。
- 录音完成：应用处于运行状态，已经结束录音，两按钮内容分别为“录音”和“播放”，顶部胶带轮盘静止，由于已完成录音操作，“录音”功能不可用，“播放”功能可用。
- 播放中：应用处于运行状态，已经完成录音，两按钮内容分别为“录音”和“暂停”，顶部胶带轮盘转动，由于已经完成了录音操作，“录音”功能不可用，“播放”功能可用。
- 暂停中：应用处于运行状态，播放声音被暂停，两按钮内容分别为“录音”和“播放”且“录音”功能不可用。
- 播放完成：应用处于运行状态，播放声音完成，两按钮分别为“录音”和“播放”，点击“播放”按钮将重新播放声音。

下面将根据以上 6 个状态来为界面加入交互。

10.8.3 界面交互的实现

根据 10.8.2 小节列出的 6 个状态，需要分别实现两个按钮被点击时的响应函数 onRecording()和 onRecordPlay()。此外还需要控制顶部胶带轮盘图片转动和静止的函数 recordRun()和 recordStop()，以及设置播放进度和录音时间的函数 setBar()和 setTime()。

新建文件 do.js 并在 index.html（范例 10-7）的第 7 行插入代码引入 JavaScript 脚本文件：

```
<script type="text/javascript" charset="utf-8" src="js/do.js "></script>
```

其中的具体内容如范例 10-9 所示。

【范例 10-9】界面实现之 do.js 部分。

```
01  var applicationStatus = 0;
02  var run_ID = 0;
03  var angle = 0;
04  // 获取录音的总长度，小数，单位为秒
05  var curTime = 0;
06  // 当前播放时间，小数，单位为秒
07  var nowTime = 0;
08  // 使用计数器获取的录音总长度，整数，单位为秒
09  var culTime =0;
10  // 为胶带轮盘转动设定计时器
11  function recordRun() {
12      run_ID = setInterval( changeAngle,50);
13  }
14  // 改变胶带轮盘转动的角度
15  function changeAngle() {
16      document.getElementById("pic").css({"transform":"rotate("+angle+"deg)",
17              "-webkit-transform":"rotate("+angle+"deg)"});
18      angle = angle + 1;
19      alert(angle);
20  }
21  // 停止胶带轮盘转动
22  function recordStop() {
23      clearInterval(run_ID);
24  }
25  //开始和结束录音
26  function onRecording() {
```

```
27      switch ( applicationStatus)
28      {
29          case 0:
30              // 未录音，点击按钮开始录音
31              applicationStatus = 1;
32              document.getElementById("button_left").innerHTML="停止";
33              recordRun();
34              break;
35          case 1:
36              // 正在录音中，点击按钮结束录音
37              applicationStatus = 2;
38              document.getElementById("button_left").innerHTML="录音";
39              recordStop();
40              break;
41          default:
42              // 其他状态，录音已经完成，按钮不可用
43              alert("按钮不可用！");
44      }
45  }
46  // 开始和结束播放
47  function onRecordPlay() {
48      switch ( applicationStatus) {
49          case 2:
50              // 录音完成，点击按钮开始播放
51              applicationStatus = 3;
52              document.getElementById("button_right").innerHTML="暂停";
53              recordRun();
54              break;
55          case 3:
56              // 播放中，点击按钮暂停
57              applicationStatus = 4;
58              document.getElementById("button_right").innerHTML="播放";
59              recordStop();
60              break;
61          case 4:
62              // 暂停中，点击按钮播放
63              applicationStatus = 3;
64              document.getElementById("button_right").innerHTML="暂停";
65              recordRun();
```

```
66              break;
67          case 5:
68              // 播放完成，点击按钮开始播放
69              applicationStatus = 3;
70              document.getElementById("button_right").innerHTML="暂停";
71              recordRun();
72              break;
73          default:
74              // 其他状态，按钮不可用
75              alert("按钮不可用！");
76      }
77  }
78  // 在播放时，设置进度条以及数字时间的进度
79  function setBar() {
80      var wid = 300 * nowTime / curTime;
81      document.getElementById("bar").style.width = wid + px;
82      document.getElementById("bar").innerHTML = nowTime + "/" + curTime +
"S";
83  }
84  // 在录音时，用来设置显示数字时间
85  function setTime() {
86      document.getElementById("bar").innerHTML = culTime + "/--";
87  }
```

在第 1 行声明了一个变量 applicationStatus，用它的值来记录当前录音或者是播放的状态，在函数 onRecording()（第 26~45 行）和函数 onRecordPlay()（第 47~77 行）中都是依靠此变量来判断接下来要进行何种操作的。

第 3 行中的变量 angle 用来记录胶带转盘转动的角度，在第 15~20 行的函数 changeAngle()中对它进行引用。该函数每运行一次，胶带转盘都会转动 1°，由于 setInterval()的作用，只要不对它进行停止操作，胶带转盘将一直转动下去。



可以通过修改 setInterval()的时间参数来设置转盘转动的速度。通过动画来向用户展示应用所处的状态是一种常用而且能有效提升用户体验的方法，在实际开发中还可设置在录音和播放时转盘分别向不同方向转动来加以区分。

第 79~87 行中定义了两个函数，分别用来在录制声音和播放时设置进度条的进度，由于在录制声音时，根本不知道要录制多久因此也无法使用进度条，只能利用数字显示已经录制的秒数。当声音被播放时，则可以利用 Media 对象获取音频的总长度和当前的播放进度。而之前的 index.html 文件中，id 值为 bar 的 div 元素被加入了绿色的背景，通过调整它的宽度就

可以形成进度条一点一点增长的效果了。

此外，目前一些函数还没有完成，比如 onRecordPlay()和 onRecording()还没有开始对 Cordova 播放和录制声音的方法加以调用，这些在下一小节将会补充完整。

10.8.4　录音和播放功能的实现

以上已经实现了该项目的界面以及交互功能，但是最本质的录音功能却还迟迟没有实现。观察范例 10-7 的代码，会发现第 10 行有一句：“//此处加入 JavaScript 代码”，原来笔者早就做出了准备。

本小节就要在这个位置插入用来实现录音以及播放功能的函数，如范例 10-10 所示。

【范例 10-10】实现录音播放功能。

```
01  // 设置录音保存文件
02  var mysrc = "myrecording.mp3";
03  // 新建 Media 对象
04  var myrec = new Media(mysrc, onSuccess, onError);
05  // 设置录音计时计数器
06  var recInterval = 0;
07  // 设置录音播放计数器
08  var mediaTimer = 0;
09  // 创建 Media 对象成功
10  function onSuccess() {
11      console.log("recordAudio():Audio Success");
12  }
13  // 创建 Media 对象失败
14  function onError(error) {
15     alert('code: '    + error.code    + '\n' +
16          'message: ' + error.message + '\n');
17  }
18  // 开始录音
19  function recordStart() {
20      // 开始录音
21      myrec.startRecord();
22      // 设置计数器用来计算录音时间
23      recInterval = setInterval(function() {
24          culTime = culTime + 1;
25          // 调用 setTime()函数显示录音时间
26        setTime();
27     }, 1000);
```

```
28  }
29  // 结束录音
30  function recordFinish() {
31      // 停止录音
32      myrec.stopRecord();
33      // 结束计数器
34      clearInterval(recInterval);
35      curTime = myrec.getDuration();
36  }
37  // 播放录音
38  function recordPlay() {
39      if(myrec) {
40          myrec.play();
41          mediaTimer = setInterval(function() {
42            // 获取播放进度
43            myrec.getCurrentPosition(
44            // 成功获取播放进度
45            function(position) {
46                  // 将获取的播放进度存放在变量 nowTime 中
47                  nowTime = position;
48                  // 设置进度条显示播放进度
49                  setBar();
50            },
51            // 获取播放进度失败
52            function(e) {
53                  // 不进行操作
54            }
55            );
56        }, 1000);
57      }
58  }
59  // 暂停播放录音
60  function recordPause() {
61      // 暂停播放
62      myrec.pause();
63      // 清除计数器计时
64      clearInterval(mediaTimer);
65  }
```

至此，该应用所需要的录音和播放功能也已经实现了，代码中新定义了 4 个变量，它们的作用可以根据代码中的注释进行理解（第 1~8 行）。然后分别定义了用来实现对音频进行录制、结束录制、播放和暂停的函数，在这些函数中会用到新定义的 4 个变量。



由于变量作用域的关系，这些变量必须在函数外进行定义。

至于具体的内容在本章之前几节中都已经有过介绍，希望读者根据注释自行去理解代码的内容，也当作是一种复习。

10.8.5 最终的组合

也许有性急的读者已经迫不及待地将程序“敲”好放在虚拟机或者实体机中去运行了，却发现完全无法录音和播放。当年笔者按照书本自学数据结构时也遇到过这样的糗事，代码都“敲”进去了，但 Visual C++却一个劲地报错。虽然各个功能需要的函数都写好了，可是没经过组合又怎么能发挥作用呢？

所以尽管心急，还是要老老实实地回到范例 10-9，对 do.js 做一些简单的修改。这次的修改主要是针对 onRecording()和 onRecordPlay()两个函数进行的。

【范例 10-11】修改后的函数：onRecording()和 onRecordPlay()。

```
01 //开始和结束录音
02 function onRecording() {
03     switch ( applicationStatus)
04     {
05         case 0:
06             // 未录音，点击按钮开始录音
07             applicationStatus = 1;
08             document.getElementById("button_left").innerHTML="停止";
09             recordRun();
10             // 此处加入录音功能
11             recordStart();
12             break;
13         case 1:
14             // 正在录音中，点击按钮结束录音
15             applicationStatus = 2;
16             document.getElementById("button_left").innerHTML="录音";
17             recordStop();
18             // 加入录音结束功能
19             recordFinish();
20             break;
21         default:
```

```
22                // 其他状态，录音已经完成，按钮不可用
23                alert("按钮不可用！");
24        }
25    }
26    // 开始和结束播放
27    function onRecordPlay() {
28        switch ( applicationStatus) {
29            case 2:
30                // 录音完成，点击按钮开始播放
31                applicationStatus = 3;
32                document.getElementById("button_right").innerHTML="暂停";
33                recordRun();
34                // 加入播放功能
35                recordPlay();
36                break;
37            case 3:
38                // 播放中，点击按钮暂停
39                applicationStatus = 4;
40                document.getElementById("button_right").innerHTML="播放";
41                recordStop();
42                // 加入播放暂停功能
43                recordPause();
44                break;
45            case 4:
46                // 暂停中，点击按钮播放
47                applicationStatus = 3;
48                document.getElementById("button_right").innerHTML="暂停";
49                recordRun();
50                // 加入播放功能
51                recordPlay();
52                break;
53            case 5:
54                // 播放完成，点击按钮开始播放
55                applicationStatus = 3;
56                document.getElementById("button_right").innerHTML="暂停";
57                recordRun();
58                // 加入播放功能
59                recordPlay();
60                break;
```

```
61          default:
62              // 其他状态，按钮不可用
63              alert("按钮不可用！");
64      }
65  }
```

在范例的第 11、19、35、43、51 和 59 行分别加入了录音和播放功能的函数，使得整个程序完整起来，再运行就能够实现最初设计的功能了。



也许有读者认为按照笔者最初介绍的 Cordova 知识点时采用的习惯，应该先写好 Cordova 相关的函数（比如本章介绍的录音和播放功能的框架）才对。但是在实际开发中，这种想法却是不合适的。虽然使用 Cordova 无论是编译还是调试都非常方便，但是毕竟不如在浏览器中直接测试来得方便。因此在开发“大型”项目时，应将程序本身界面以及交互所需要的函数完成，并在本机浏览器上测试通过，之后再去考虑具体功能的实现。这样才能保证会有最高的生产效率，将利益最大化。

虽然这个“录音机”已经可以使用了，但它还只是一个模型，有许多不完善的地方，比如它只能录制一段声音并且没有保存的功能。这很大程度上是因为还有后续章节的一些内容需要学习所致，有兴趣的读者可以根据后续的内容想方设法将它完善起来。

10.9 小结

本章学习了使用 Cordova 中的 Media 对声音媒体文件进行操作的方法，并实现了一个简单的录音机应用。通过本章的学习，读者应该已经有能力去独立实现一些应用，比如网络播放器、录音机等。但是，许多知识却并不是仅靠一本书就能介绍清楚的，比如 release()方法的使用以及媒体文件对设备硬件的占用等，这都需要在实际的开发中去体会。

第 11 章

Cordova中的文件操作

本章将介绍 Cordova 中最为复杂的一类对象——文件操作类对象，这是一组用于读取、修改和写入文件信息的对象和 API。与前几章所介绍的内容不同，以往 Cordova 通常是将一类操作完全封装在同一个对象中，比如 Media 对象包含了全部对声音操作的方法，但是 Cordova 对文件的操作方法却被封装在了许多不同的对象中，这就导致了学习难度的增加。本章将会更加着重于对基础知识的介绍，而不是简单的实例，以便读者理解 Cordova 的文件操作。

本章主要内容包括：

- Cordova 的系统目录结构，包括 DirectoryEntry 和 DirectoryReader 类的用法和介绍。
- Cordova 的文件系统，包括 File、FileEntry 等对象以及它们对文件进行读写的方法和概念。
- 对 Cordova 文件操作类中各个方法所使用参数的介绍。

11.1 使用 FileReader 读取文件

FileReader 类是 Cordova 文件系统中用来读取文件信息的一类操作。在这些操作中，文件中的信息将以字符串或者 Base 64 编码的形式被读出，开发者可以自定义监听器来处理读取的数据。

FileReader 类中的方法如表 11-1 所示。

表 11-1 FileReader 类中的方法

方法名	作用
abort	终止对文件的读操作
readAsDataURL	以 Base 64 编码的形式返回数据
readAsText	以纯文本的形式返回读取的数据
readAsBinaryString	以二进制形式返回读取的文本
readAsArrayBuffer	将读取的数据以数组的形式返回

在表 11-2 中是表 11-1 中的方法会用到的一些参数。

表 11-2　使用 FileReader 类进行操作时会用到的参数

参数名	说明
readyState	当前文件读取操作所处的状态，只能是以下三个值之一：EMPTY、LOADING 和 DONE，分别代表未读取、读取中和读取完毕
result	从文件中读取的内容，类型为字符串
error	读操作遇到错误时会包含错误信息
onloadstart	成功开始文件读取时调用的回调函数的名称
onpregress	读取过程中的回调函数，用于在读取中获取读取进度，显示进度条等操作
onload	读操作完成后调用的回调函数，一般在此时对读取的数据进行处理
onabort	当读操作被终止时会被调用的回调函数
onerror	当读操作失败后会被调用的回调函数
onloadend	无论读操作进行成功或者失败，在请求完成后都会被调用的回调函数

从上面两个表格来看，实际上 FileReader 类的使用与之前学习的其他类的使用没有太大的差别，因为都是遵循 W3C 规范制定的。那么接下来就可以尝试利用它读取文件了。具体实现代码如范例 11-1 所示。

【范例 11-1】利用 FileReader 类读取一个文件。

首先我们进入虚拟机中 app 的沙盒中，在 tmp 文件夹中放一个 readme.txt，内容就是“这是一个测试 txt”，如图 11-1 所示。



图 11-1　在 app 沙盒中添加 txt 文件

```
01  <!DOCTYPE HTML>
02  <html>
03  <head>
04      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05      <title>利用 FileReader 类读取一个文件</title>
06      <script type="text/javascript" charset="utf-8" src="js/cordova.js">
</script>
07      <script type="text/javascript" charset="utf-8">
```

```
08            // 设置设备加载的监听器
09          document.addEventListener("deviceready", onDeviceReady, false);
10        // 设备加载完毕
11        function onDeviceReady() {
12          window.requestFileSystem(window.TEMPORARY, 0, gotFS, fail);
13        }
14        // 读取文件 readme.txt
15        function gotFS(fileSystem) {
16          fileSystem.root.getFile("readme.txt", null, gotFileEntry, fail);
17        }
18        // 获取文件句柄
19        function gotFileEntry(fileEntry) {
20            // 读取文件句柄
21          fileEntry.file(gotFile, fail);
22        }
23        // 读取文件内容
24        function gotFile(file){
25          //readDataUrl(file);
26          readAsText(file);
27        }
28        // 以 Base 64形式读取文件
29      function readDataUrl(file) {
30          var reader = new FileReader();
31          reader.onloadend = function(evt) {
32              // 下面两行用来以 Base 64形式显示读取的文件的，此处先注释掉，读者可自行查看
结果
33              // alert("Read as data URL");
34              // alert(evt.target.result);
35          };
36          reader.readAsDataURL(file);
37        }
38        // 以文本方式读取文件
39        function readAsText(file) {
40          var reader = new FileReader();
41          reader.onloadend = function(evt) {
42              alert("以文本形式读取");
43              alert(evt.target.result);
44          };
45          reader.readAsText(file);
```

```
46          }
47          // 读取文件失败
48          function fail(evt) {
49              alert(evt.target.error.code);
50          }
51          </script>
52      </head>
53  <body>
54      <h1>利用 FileReader 类读取一个文件</h1>
55  </body>
56  </html>
```

运行之后可以看到，结果如图 11-2 所示。



有不少读者在运行了本范例之后发现完全不是这样的结果，因为本范例的作用是将读取的 readme.txt 中的内容显示出来。所以，在进行实验之前一定要保证在应用的根目录下存在 readme.txt 这个文件。

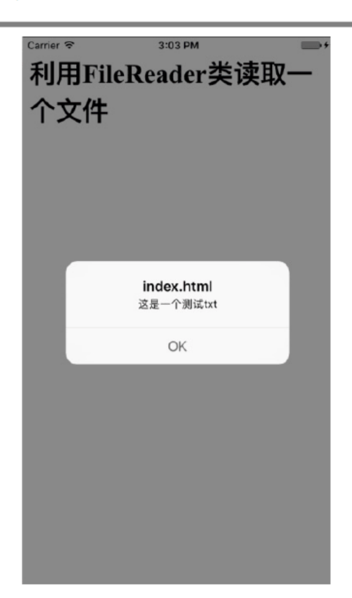


图 11-2　将读取的文件内容输出

笔者在学习 Cordova 文件系统时，曾经一度由于它的复杂性而产生一定的畏惧心理，将这部分放到了最后才学习。在对它有了一知半解时又大言不惭地表示它非常简单。那么 Cordova 文件操作到底难不难呢？接下来对范例的讲解也许能带来答案。

当设备完成准备后，执行命令 window.requestFileSystem(window.TEMPORARY, 0, gotFS, fail)开始读取文件。

没想到才进行了一步就遇上了一个大难题，window.requestFileSystem()是什么东西，之前

完全没有学习过啊。因为这个方法并不是 Cordova 中的，而是在 HTML5 中用来请求对沙盒文件系统操作权限的一个函数，简单点说就是只有执行了这个命令才能操作系统中的文件（要不文件安全怎么保障呢）。

该函数的原形如下：

```
window.requestFileSystem(type, size, successCallback, opt_errorCallback)
```

其中使用了 4 个参数，这些参数的说明如下：

- type：用于声明存储文件的类型，分为持久型的还是非持久型的，分别为 LocalFileSystem.PERSISTENT 和 window.TEMPORARY。定义为非持久型的数据将会在页面被关闭后被浏览器自动删除。当然，由于本例中只是对文件进行读取，所以使用哪个选项与结果关系不大。
- size：定义了可由用户操纵文件的总的大小，为 0 则表示无限。
- successCallback：当请求成功后执行的回调函数，本例将在此函数中进行下一步的操作。
- opt_errorCallback：请求失败后执行的回调函数。



许多开发者忽略了该函数请求的权限为一个“沙盒”文件系统的操作权限，沙盒一词就说明了该权限所能够操纵的文件必然有一定的局限性。比如一个应用无法对另一个应用所申请的文件系统进行操作，这种设计为 HTML 5 的安全性提供了一定的保证。

第 16 行是在权限请求成功之后进行的操作，使用了方法 fileSystem.root.getFile()，fileSystem 是系统在请求权限成功时返回的一个 FileSystem 类型的对象，它包含了文件默认路径等一系列信息。本例使用它来实现对文件 readme.txt 的读取。读取成功后将进入自定义函数 gotFileEntry 中进行下一步的操作。

gotFileEntry()函数位于范例的第 19~22 行，该操作将默认获取一个类型为 FileEntry 类型的对象 fileEntry。该函数利用它的 file()方法来获取文件 readme.txt 的句柄。在获取成功后将进入另一个自定义函数 gotFile。在 gotFile()函数中又遇上了一个新的变量 file，本例将执行函数 readAsText(file)来对它进行读取，见范例第 26 行所示。

readAsText()函数在范例的第 39~46 行被定义，直到这时才算是真正地用到了 FileReader 类（如第 40 行所示）。

这下就不难看出来了，其实 Cordova 中的文件系统并不难，但是由于在使用时要牵扯到的知识点太多以至于无法下手。所以读者一定要注意把前后的知识点连贯起来，虽然有许多类，但是读者在实际开发中一定要形成这是一个系列的概念才行。

11.2 使用 FileWriter 编写文件

上一节介绍的 FileReader 是一个用来对文件进行读操作的类，本节将要介绍的是一个用来对文件进行写操作的类，它的名字叫 FileWriter。

与 FileReader 一样，它也封装了若干用来对文件进行操作的方法，如表 11-3 所示。

表 11-3　FileWriter 对象中的方法

方法名	说明
abort	终止对文件的写入操作
seek	移动文件指针到相应的位置
truncate	按照指定长度截断文件
write	向文件中写入数据
append	在文件的结尾处写入数据

在使用这些方法时也要用到一些对象作为参数，如表 11-4 所示。

表 11-4　使用 FileWriter 进行文件写操作时会用到的参数

参数名	说明
readyState	在对文件进行写操作时，写入器所处的状态，有三组值可选，包括 INIT、WRITING、DONE
filename	要被写入文件的文件名
length	被写入文件的文件总长度
Position	文件指针的当前位置
error	错误信息
onwritestart	开始执行写操作时被调用的回调函数的函数名
onprogress	在执行写操作时执行的回调函数，用来更新写操作的操作进度
onwrite	写入文件成功后的回调函数
onabort	在写操作被终止时执行的回调函数的函数名
onerror	当写操作过程中发生错误所调用的回调函数的函数名
onwriteened	当写操作完成时执行的回调函数

下面介绍 FileWriter 类中的主要方法。

1. seek()方法

此方法用于修改文件写入的位置，使用方法如下：

```
01  function win(writer) {
02      // 快速将文件指针指向文件的尾部
03      writer.seek(writer.length);
04  };
05  var fail = function(evt) {
06      console.log(error.code);
07  };
08  entry.createWriter(win, fail);
```

在使用 seek()方法之前首先要获得一个 FileEntry 类型的对象（由于这个对象在之后才会进行介绍，此处不做深究）。第 8 行的 entry 正是一个这样的对象，调用它的 createWriter()方法，将执行到自定义函数 win 中。

在这里 writer 对象就是一个 FileWriter 类型的实例，在第 3 行调用了它的 seek()方法，用来将位置指针移动到文件的结尾处。也就是说，在使用该方法将位置指针移动到文件结尾后，再向文件中写入数据，新写入的数据将被加入到文件的结尾处。

这里以一个简单的文本文件为例，解释 seek()方法的使用，假设现在有一个文本文件，其中的内容为 phone，假如对它执行了操作 writer.seek(writer.length)后，再向文件中写入文本 GaP，过程如图 11-3 所示。

2. truncate()方法

此方法用于在指定的长度处将文件截断，使用方法如下：

```
function win(writer) {
    writer.truncate(5);
}
var fail = function(evt) {
    console.log(error.code);
}
entry.createWriter(win, fail);
```

比如，将含有内容为 Cordova 的文件进行上面代码中的操作后，文件中的内容将只剩下 Phone。

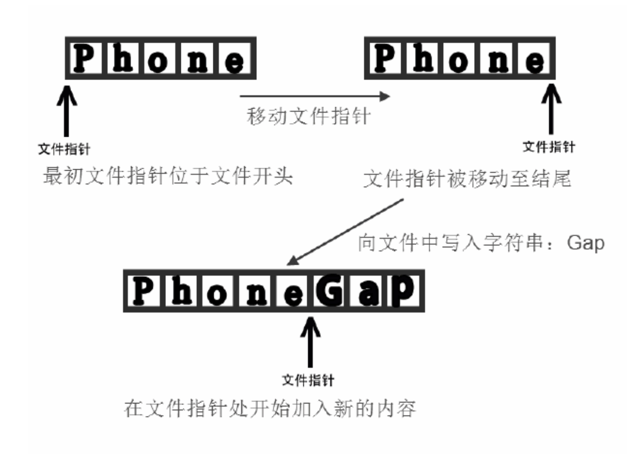


图 11-3　移动文件指针在文件结尾处写入字符串的图例



有许多读者认为该方法实在是有些鸡肋之嫌，首先它与 JavaScript 自带的那些字处理方法相比实在是不够强大和方便，其次是 Cordova 开发者有许多是由 Web 开发者转行而来的，这使得他们对 JavaScript 的熟悉程度要远高于 Cordova。但是事实上，truncate()方法自然有它的方便之处，比如使用 truncate(0)就隐含了一个简单的清空操作，而在实际使用中也可以用 truncate(MAXSIZE)来限制文件的最大长度以保证程序的健壮性。

3. write()方法

在图 11-3 所示的过程中，还有一个步骤是向文件中写入内容，而这一步骤显然仅通过 truncate()方法是无法完成的。这时就需要用到 FileWriter 类中的另一个常用的方法：wrire()方法。

此方法用于向文件的指针位置写入数据，用法如下：

```
function win(writer) {
 writer.onwrite = function(evt) {
  console.log("write success");
 };
writer.write("some sample text");
};
var fail = function(evt) {
  console.log(error.code);
};
entry.createWriter(win, fail);
```

通过此方法就能够将字符串 some sample text 写入到文件指针的位置。

在笔者学习本部分内容时，曾经有一个疑问，如果一个文件中原有的内容是

1234567890，此时的文件指针位于 1 和 2 之间，此时文件指针的位置如图 11-4 所示。那如果在此时向文件中写入一个字符串 Cordova，会怎么样呢？

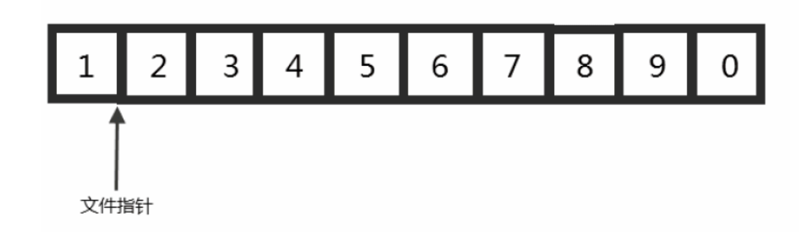


图 11-4　此时的文件指针位置示意

实践证明，此时向文件中写入内容 Cordova 后文件中的内容变成了“1Cordova90”，而不是笔者之前认为的“1Cordova234567890”，即，利用 write()方法向文件中写入内容会覆盖原有的内容。

4. abort()方法

该方法的作用是在向文件中写入内容的过程中终止系统的“写”操作。

接下来将给出一个综合使用 FileWriter 类进行文件写操作的例子，代码如范例 11-2 所示。

【范例 11-2】利用 FileWriter 类进行文件的写操作。

```
01  <!DOCTYPE HTML>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>利用 FileWriter 类进行文件的写操作</title>
06  <script type="text/javascript" charset="utf-8" src="js/cordova.js">
</script>
07  <script type="text/javascript" charset="utf-8">
08  // 声明一个设备触发器
09  document.addEventListener("deviceready", onDeviceReady, false);
10  // Cordova 加载完毕
11  function onDeviceReady() {
12      window.requestFileSystem(window.TEMPORARY, 0, gotFS, fail);
13  }
14  // 获取文件系统
15  function gotFS(fileSystem) {
16      fileSystem.root.getFile("readme.txt", null, gotFileEntry, fail);
17  }
```

```
18  function gotFileEntry(fileEntry) {        // 获取文件入口
19      fileEntry.createWriter(gotFileWriter, fail);
20  }
21  // 进行"写"操作
22  function gotFileWriter(writer) {
23      writer.onwrite = function(evt) {
24          console.log("write success");
25      };
26      writer.write("some sample text");
27      // 文件当前内容是 some sample text
28      writer.truncate(11);
29      // 文件当前内容是 some sample
30      writer.seek(4);
31      // 文件当前内容依然是 some sample，但是文件的指针位于 some 的 e 之后
32      writer.write(" different text");
33      // 文件的当前内容是 some different text
34  }
35  function fail(error) {
36      console.log(error.code);
37  }
38  </script>
39  </head>
40  <body>
41      <h1>利用 FileWriter 类进行文件的写操作</h1>
42  </body>
43  </html>
```

读者可以运行本程序来查看结果。由于本例运行的结果无法直接用图片来给出，因此笔者将不再给出结果，而分析代码的第 22~34 行的每一步操作来将运行后的结果告知读者。

在第 22 行的内容中，writer 是一个 FileWriter 类型的对象，此时文件内容为空。在第 23~25 行处的 onwrite()函数中利用 LogCat 窗口输出内容 write success，表示能够成功执行写操作。再在第 26 行使用 write()方法，向文件中写入字符串 some sample text，文件指针位于文件的结尾处，即字母 t 之后。



可以理解为 write()方法在向文件中写入内容时是一个个字符依次写入的，为了保证能够让下一个字符顺利地写在刚刚写入字符的后面，以保证写入字符串时顺序正确，需要将文件指针后移一位。因此，当执行完写操作之后，文件指针就被移动到了新写入字符串的结尾处。

接下来使用 truncate()方法将文件截断，仅保留文件中前 11 位的内容（范例第 28 行所

示），此时文件中的内容为 some sample，文件指针依然位于字符串的结尾。



如果文件指针位于文件被截断的部分字符串位置，那么截断后的文件指针将位于新文本的结尾处。

在第 30 行，使用了 seek()方法来移动文件指针到第 4 个字符后，此时文件指针位于 some 的 e 之后，再使用 write()方法向文件中写入内容（第 32 行）。原本的 sample 字符被替换，此时文件中的内容为 some different text，文件指针再次位于文本的结尾处。

11.3 使用 FileSystem 获取文件系统信息

前面两节分别介绍了 FileRead 类和 FileWrite 类的使用，这两个类封装了用于实现对文件读写的操作，是在文件操作中最容易被“看见”的操作。但是在文件系统中，并不是所有的操作都是可被“看见”的，比如本节要介绍的 FileSystem 类。



由于 FileSystem 本身也被翻译为文件系统，而 FileSystem 类本身又是 Cordova 文件系统的一部分，因此容易被混淆，所以读者每当遇到“文件系统”这 4 个字时要仔细推敲，根据语境分析它的具体含义。

FileSystem 类封装了当前文件系统的信息，它有两个属性 name 和 root。其中 name 属性在公开的文件系统中是唯一的，而 root 属性则包含了一个代表当前系统所在根目录的 DirectoryEntry 对象。

【范例 11-3】利用 FileSystem 获取文件系统信息。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <title>利用 FileSystem 获取文件系统信息</title>
05  <script type="text/javascript" charset="utf-8" src="js/cordova.js">
</script>
06  <script type="text/javascript" charset="utf-8">
07      // 设置监听器，等待加载 Cordova
08      document.addEventListener("deviceready", onDeviceReady, false);
09      // Cordova 加载完毕
10      function onDeviceReady() {
11          // 设备加载完毕，申请访问文件系统的权限
12          window.requestFileSystem(LocalFileSystem.PERSISTENT,0,onFileSystemSuccess,
fail);
```

```
13          }
14          // 获取文件系统信息
15          function onFileSystemSuccess(fileSystem) {
16              // 在日志中输出文件系统信息
17              alert("文件系统名称："+fileSystem.name);
18              //alert("根目录名称："+fileSystem.root.name);
19          }
20          // 获取文件信息失败
21          function fail(evt) {
22              alert(evt.target.error.code);
23          }
24      </script>
25      </head>
26      <body>
27          <h1>利用 FileSystem 获取文件系统信息</h1>
28      </body>
29      </html>
```

运行之后，结果如图 11-5 所示。

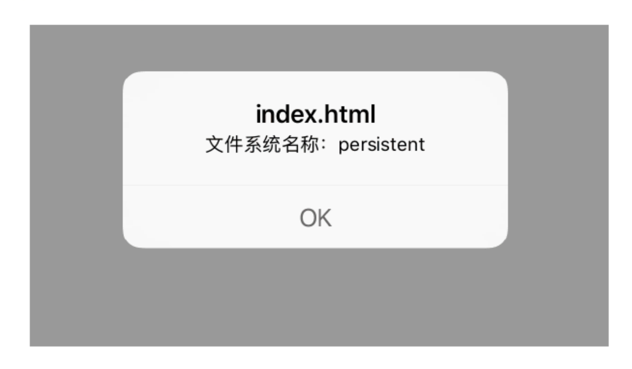


图 11-5　窗口中输出的文件系统信息

在 FileSystem 类中，文件系统的相关信息分别被封装在 name 和 root 两个属性中，name 本身就是一个字符串型的对象，用来表示文件系统的名称。

11.4 FileEntry 类简介

在上一节中曾经提到过，在 FileSystem 类中封装了一个名为 root 的属性，它是一个 FileEntry 对象，本节将会对它进行详细介绍。

FileEntry 实际上代表文件系统中的一个文件，以起到“文件入口”的作用，W3C 目录和

系统规范对其进行了详细的定义。但是比较遗憾的是，FileEntry 中的一个属性 filesystem 虽然在 W3C 中被定义了，可是在 Cordova 中却是一个“无意义”的内容，因为它并不支持。

表 11-5 和表 11-6 中分别是 FileEntry 类中的属性和方法。

表 11-5　FileEntry 类中的属性

名称	描述
isFile	该属性总是返回 true
isDirectory	总是返回 false
name	FileEntry 的名称，但是不包括它们的路径
fullPath	从文件系统的根目录到 FileEntry 的完整路径
filesystem	FileEntry 所在的文件系统名称，但是该属性仅仅被 W3C 定义了却不被 Cordova 支持

表 11-6　FileEntry 类中的方法

方法名	描述
getMetadata	获取文件的元数据
moveTo	移动一个文件到不同的位置
copyTo	复制一个文件到不同的位置
toURL	返回一个可定为文件位置的 URL 对象
remove	删除一个文件
getParent	查找父级目录
createWriter	创建一个 FileWriter 对象
file	创建一个包含文件属性的 File 对象

通过表格可以看出在 FileEntry 类中封装了不少方法，因此接下来要做的就是学习每个方法的用法。

1. getMetadata()方法

该方法主要用来查找文件中的元数据，使用方法如下：

```
function success(metadata) {
    console.log("Last Modified:" + metadata.modificationTime);
}
function fail(error) {
    alert(error.code);
}
// 请求这个条目的元数据对象
entry.getMetadata(success, fail);
```

当该方法执行成功后，将会在回调函数 success()中返回一个元数据类型，可以在其中对该数据进行处理，属于很少被使用到的方法之一。



元数据类型是一个用来表示文件状态的类，在本章的后续内容中专门对它进行介绍。

2. moveTo()方法

该方法可以移动一个文件到其他位置，类似于“剪切”+“粘贴”的操作。其使用方法如下：

```
function success(entry) {
    console.log("New Path: " + entry.fullPath);
}
function fail(error) {
    alert(error.code);
}
function moveFile(entry) {
    // 这里声明一个新路径
    var parent = document.getElementById('parent').value,
        parentEntry = new DirectoryEntry({fullPath: parent});
    // 移动文件到一个新目录，并将其重命名
    entry.moveTo(parentEntry, "newFile.txt", success, fail);
}
```

在使用 moveTo()方法时，还要注意以下三点：

- 当在同一个文件夹中对文件进行移动时（就像先剪切但不切换目录直接进行粘贴操作一样）会导致操作失败。
- 移动文件到一个被占用的路径中会遭遇失败。
- 当尝试将一个文件移动到另一个路径下时，若该路径下有与该文件同名文件，则原文件被移来的文件所替换。

3. copyTo()方法

该方法与 moveTo()方法非常类似，类似于“复制”+“粘贴”操作，具体使用方法如下：

```
function win(entry) {
    console.log("New Path: " + entry.fullPath);
}
function fail(error) {
    alert(error.code);
}
function copyFile(entry) {
    // 声明新的目录
    var parent = document.getElementById('parent').value,
```

```
        parentEntry = new DirectoryEntry({fullPath: parent});
    // 复制文件到一个新的目录，并将其重命名
    entry.copyTo(parentEntry, "file.copy", success, fail);
}
```

4. toURI()方法

使用该方法将返回一个可用于定位文件路径的 URI 对象，使用方法如下：

```
var uri = entry.toURI();
console.log(uri);
```



该方法虽然使用简单，但是确实非常实用，尤其是在执行 copyTo、moveTo 等方法前，需要用此方法来获取一些与操作有关的数据。

5. remove()方法

该方法用于删除一个文件，使用方法如下：

```
function success(entry) {
    console.log("Removal succeeded");
}
function fail(error) {
    alert('Error removing file: ' + error.code);
}
// 移除该文件
entry.remove(success, fail);
```

6. getParent()方法

该方法能够返回当前文件的父级目录，使用方法如下：

```
function success(parent) {
    console.log("Parent Name: " + parent.name);
}
function fail(error) {
    alert(error.code);
}
// 获得父级 DirectoryEntry 对象
entry.getParent(success, fail);
```

该方法返回的父级目录是一个 DirectoryEntry 类的对象，可以通过它来跳转到当前文件的父目录中去。尤其要注意在该方法中即使获取目录失败，其回调函数中仍旧会返回一个空的 DirectoryEntry 对象。

7. createWriter()方法

该方法在本章的范例 11-2 中曾经被使用过，它可以返回一个“可操作的”FileWriter 对象，从而由对文件目录的操作转移到对文件的操作（读写）中来，它的使用方法如下：

```
function success(writer) {
    writer.write("Some text to the file");
}
function fail(error) {
    alert(error.code);
}
// 创建一个用于写文件的 FileWriter 对象
entry.createWriter(success, fail);
```

8. file()方法

该方法可以返回一个 File 类型的对象，其中包含了当前文件的大小、名称等信息，它的使用方法如下：

```
function success(file) {
    console.log("文件大小: " + file.size);
}
function fail(error) {
    alert("Unable to retrieve file properties: " + error.code);
}
// 获得此文件的属性
entry.file(success, fail);
```

在此方法中返回的 File 类在 Cordova 中并没有被定义，因为 W3C 中已经为 JavaScript 定义了一个完整的 File 类型，其各个属性以及描述参见表 11-7。

表 11-7　File 对象中的属性

属性名称	描述
Attributes	设置或返回文件或文件夹的属性
DataCreated	返回指定文件或文件夹的创建时间
DateLastAccessed	返回最近访问文件或文件夹的创建时间
DateLastModified	返回最后修改指定文件和文件夹的日期
Drive	返回指定文件或文件夹所在驱动器的驱动器号
Name	设置或返回文件或文件夹的名称
ParentFolder	返回指定文件或文件夹的对象
Path	返回指定文件或文件夹的路径
ShortName	返回短名称
ShortPath	返回短路径
Size	对于文件，以字节为单位返回指定文件的大小 对于文件夹，以字节为单位返回其中所包含的全部文件及文件夹的大小
Type	返回文件或文件夹的信息

11.5 DirectoryEntry 类的简介

DirectoryEntry 这个类在用法和功能上与上一节介绍的 FileEntry 非常类似，它代表了文件系统中的一个目录。因为与 FileEntry 类似，认真学习过上一节内容的读者能够很快掌握 DirectoryEntry 的使用方法，在 DirectoryEntry 类中包含的属性和方法名称与 FileEntry 是非常类似的，下面将对该类中的方法进行介绍。

1. getMetadata()方法

该方法用于查找对应目录中的元数据，使用方法如下：

```
// 获取元数据成功，在日志中输出元数据内容
function success(metadata) {
    console.log("Last Modified:" + metadata.modificationTime);
}
// 获取元数据失败，输出错误代码
function fail(error) {
    alert(error.code);
}
// 请求当前目录下的所有元数据对象
entry.getMetadata(success, fail);
```

2. moveTo()方法

该方法用于移动一个目录到其他位置，但是当进行以下操作时该方法会失败：

- 将原目录移动到其子目录中。
- 将移动的目标目录设为当前目录（即将目录移动到当前位置，即使成功也相当于没有移动）。
- 将目录移动到一个被其他资源所占用的目录下（即没有相应的权限）。
- 移动到一个不存在的目录中。

该方法的使用如下：

```
// 操作成功，输出目录被移动到新的路径
function success(entry) {
    console.log("New Path:" + entry.fullPath);
}
// 操作失败，输出错误代码
function fail(error) {
    alert(error.code);
}
```

```
// 对目录进行移动操作
function moveDir(entry) {
    // 定义一个新的路径
    var parent = document.getElementById('parent').value;
    newName = document.getElementById('newName').value;
    parentEntry = new DirectoryEntry({fullPath: parent});
    // 移动目录到一个新目录，并将其重命名
    entry.moveTo(parentEntry, newName, success, fail);
}
```

3. copyTo()方法

该方法用于将目标目录复制到其他路径下，使用方法如下：

```
// 复制操作成功
function win(entry) {
    console.log("New Path:" + entry.fullPath);
}
// 操作失败
function fail(error) {
    alert(error.code);
}
// 进行复制操作
function copyDir(entry) {
    // 创立一个新的目录
    var parent = document.getElementById('parent').value,
        newName = document.getElementById('newName').value,
        parentEntry = new DirectoryEntry({fullPath: parent});
   // 复制目录到一个新的目录，并将其重命名
   entry.copyTo(parentEntry, newName, success, fail);
}
```

与 moveTo()方法类似，当想要将当前目录复制到其子目录或者在当前目录中进行复制操作时该方法会失败。

4. toURI()方法

返回当前目录所在的路径，使用方法非常简单，如下：

```
// 请求此条目的 URI
var uri = entry.toURI();
// 输出当前路径
```

```
console.log(uri);
```

5. remove()方法

用于删除当前目录，使用方法如下：

```
// 删除目录成功
function success(entry) {
    console.log("Removal succeeded");
}
// 删除目录失败
function fail(error) {
    alert('Error removing directory: ' + error.code);
}
// 删除操作
entry.remove(success, fail);
```



该方法在使用时有着不小的局限性，比如只能删除空的目录，而且笔者曾经尝试过删除该文件系统的根目录，发现无法成功。

6. getParent()方法

用于查找当前目录所在的目录，使用方法如下：

```
// 获取父级目录成功，输出目录名称
function success(parent) {
    console.log("Parent Name: " + parent.name);
}
// 获取失败
function fail(error) {
    alert('Failed to get parent directory: ' + error.code);
}
// 获得父级 DirectoryEntry 对象
entry.getParent(success, fail);
```



不要妄想使用此方法去获取文件系统根目录的父级目录，因为许多人已经尝试过并且失败了。

7. createReader()方法

该方法用于创建一个新的 DirectoryReader 对象以便读取目录中的内容，使用方法如下：

```
// 创建一个 DirectoryReader 对象
```

```
var directoryReader = entry.createReader();
```



DirectoryReader 与前面介绍的 FileReader 在使用上有着相似的地方，可以用来读取一个文件目录中的内容。

8. getDirectory()方法

创建一个新的目录，使用方法如下：

```
// 创建成功
function success(parent) {
    console.log("Parent Name:" + parent.name);
}
// 创建失败
function fail(error) {
    alert("Unable to create new directory:"+ error.code);
}
// 查找在当前目录下是否有名为 newDir 的目录存在，如果不存在则创建该目录
entry.getDirectory("newDir", {create: true, exclusive: false}, success, fail);
```

该方法有 4 个参数，第一个参数在范例中的值为 newDir，表示要创建目录的名称，可以写目录的完整路径，当仅设置目录名称时，目录被新建在当前目录下。第二个参数在范例中内容为{create: true, exclusive: false}，它是一个 Flags 类型的变量，其中包含了两个属性，用来指定在当前或者指定目录中不存在第一个参数中设置的目录时是否创建新的目录。剩下的两个参数则是用来设置创建目录成功和失败时所调用的回调函数。

9. getFile()方法

用于创建新的文件或者查询一个已经存在的文件，使用方法与 getDirectory()方法类似，具体使用如下：

```
function success(parent) {
    console.log("Parent Name:" + parent.name);
}
function fail(error) {
    alert("Unable to create new directory:"+ error.code);
}
// 检索一个已存在的目录，如果该目录不存在时则创建该目录
entry.getFile("newFile", {create: true, exclusive: false}, success, fail);
```

10. removeRecursively()方法

该方法用于删除一个目录以及目录中的所有内容，使用方法如下：

```
// 删除成功
function success(parent) {
    console.log("Remove Recursively Succeeded");
}
// 删除失败
function fail(error) {
    alert("Failed to remove directory or it's contents:" + error.code);
}
// 删除此目录及其所有内容
entry.removeRecursively(success, fail);
```

remove()方法也可以删除一个目录，但是却不能删除目录中的内容，该方法则是能够将目录以及目录下的全部内容一同删除。因此，在一般的删除操作中，用该方法比 remove()方法具有更广泛的用途。

那是不是 remove()方法就没用了呢？当然不是的，在许多应用中需要判断目录是否为空，如果为空才可以进行删除，比如一些数据的初始化操作，而这时就比较适合使用 remove()而不是 removeRecursively()。



DirectoryEntry 和 FileEntry 有着许多相似的地方，在用法上几乎是相同的，那么它们是不是可以混用呢？当然不行，因为 FileEntry 中的方法大多是针对单一文件进行操作的，而 DirectoryEntry 中的方法则大多是针对整个目录的。

11.6 使用 FileTransfer 向服务器上传文件

与传统的“单机”应用相比，在互联网时代被用户所喜闻乐见的还是一些具有更强交互性的社交类应用。在一些办公应用中为了实现数据的备份，也需要将一些文件上传到服务器进行保存。这样的话仅靠文件系统的操作就不好用了，急需一种新的有效的方法来实现将本机文件上传到网络的功能。

本节要介绍的 FileTransfer 类，可以实现从设备向服务器上传文件的操作，它只包含了一个方法：upload()。

在使用 FileTransfer 之前需要先添加插件:cordova plugin add https://github.com/apache/cordova-plugin-file-transfer.git。

【范例 11-4】使用 FileTransfer 类将本地文件上传到服务器。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <title>使用 FileTransfer 类将本地文件上传到服务器</title>
```

```
05  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
06  <script type="text/javascript" charset="utf-8" src="js/cordova.js">
    </script>
07  <script type="text/javascript" charset="utf-8">
08       // 设置设备加载完毕的触发器
09      document.addEventListener("deviceready", onDeviceReady, false);
10       function onDeviceReady() {
11       // 先从摄像头获取一张图片
12      navigator.camera.getPicture(uploadPhoto,
13                                function(message) { alert('get picture
failed'); },
14                                { quality: 50,
15                                  destinationType:
navigator.camera.DestinationType.FILE_URI,
16                                  sourceType:
navigator.camera.PictureSourceType.PHOTOLIBRARY }
17                                );
18
19      }
20       // 上传摄像头获得的图片
21      function uploadPhoto(imageURI) {
22          var options = new FileUploadOptions();
23            // 表单元素值
24          options.fileKey="file";
25            // 文件在服务器中保存所使用的名称
26          options.fileName=imageURI.substr(imageURI.lastIndexOf('/')+1);
27            // 文件格式
28          options.mimeType="image/jpeg";
29            var params = {};
30          params.value1 = "test";
31          params.value2 = "param";
32            options.params = params;
33            var ft = new FileTransfer();
34            // 使用了 upload()方法进行上传
35            ft.upload(imageURI, encodeURI("http://202.118.89.139/"), win, fail,
options);
36      }
37      function win(r) {
38          // 上传成功则可在此输出信息
```

```
39          alert("图片上传成功");
40      }
41      function fail(error) {
42          // 上传失败
43          alert("图片上传失败");
44      }
45  </script>
46  </head>
47  <body>
48      <h1>使用 FileTransfer 类将本地文件上传到服务器</h1>
49  </body>
50  </html>
```

下面是对范例代码的讲解：

在第 12~17 行，使用了 camera 对象来操纵摄像头获取一张照片，该方法在下一章会介绍，此时可简单忽略，只需知道是获取了一个图片文件即可。

第 21~36 行则是实现上传文件功能的核心代码，在第 33 行中，声明了一个新的 FileTransfer 类型的对象 ft，然后直接执行 upload()方法来实现文件的上传。该方法中一共有 5 个参数，对它们的解释如图 11-6 所示。

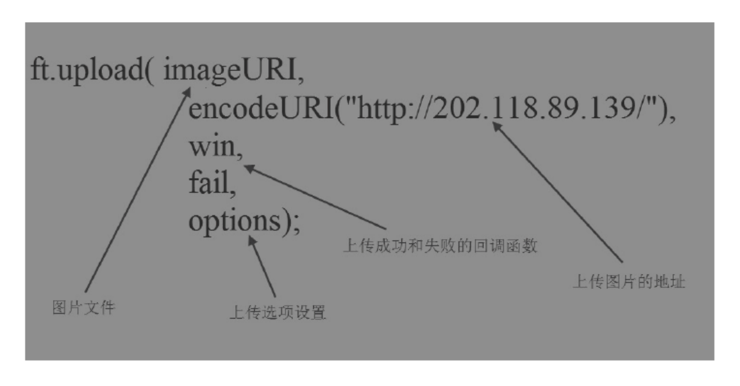


图 11-6　upload()方法的使用说明

其中，imageURI 是在通过摄像头获取图片时得到的，在范例第 12 行中可以看到 uploadPhoto 在这里被调用，而 imageURI 正是它的参数。在第二个参数 encodeURI("http://202.118.89.139/")中，http://202.118.89.139/是图片被上传的目标服务器地址，可以利用 AppServ 或者 Xampp 等软件在计算机上快速搭建。encodeURI()是 JavaScript 中定义的一个方法，用来将字符串转化为合法的 URL 对象。win 和 fail 分别是在 upload()方法执行后对执行结果成功和失败时调用的回调函数名称，这种方法在 Cordova 中非常常见。最后一个参数 options 非常重要，它是一个 FileUploadOptions 类型的对象，用来指定文件被上传时的各种参数，表 11-8 中是 FileUploadOptions 各个属性的名称及含义。

表 11-8　FileUploadOptions 类的属性

属性名称	说明
fileKey	表单元素的 name 值，默认值为 file
filename	文件被上传到服务器后被保存的文件名，默认为 file.jpg
mimeType	被上传文件的 mime 类型，默认为 image/jpeg
params	通过 http 请求发送给服务器的一系列键值对
chunkedMode	数据是否分块上传，默认为 true

从表 11-8 中可以看出，范例第 22~33 行是在执行上传操作前先设置了上传文件属性的，即以照片在设备中保存的默认名称上传到服务器中。

11.7 其他与文件系统相关的类

本章介绍了 Cordova 中关于文件操作的好多类，而且还有好几个类没有介绍，不知道各位读者对此有什么看法，笔者在学习这部分内容时研究了好久才大概明白了这些类之间的关系，最终还是靠实践才渐渐摸索出了一些门道。为此，本节介绍 Cordova 文件系统中剩余的几个类，同时做一个类似“导学”的介绍，使读者能够对于怎样将这些类联系起来有一个清晰的认识。

有一种叫做“密室逃生”的室内探险游戏，一般会将几个“冒险者”关在一个封闭的小房间内，然后游戏的玩家可能会在“密室”的各个角落找到一些诸如钥匙、谜题或者上着锁的箱子之类的物件，然后根据提示又能找到新的钥匙或者箱子等，一环套着一环。图 11-7 是一个线上密室逃生游戏的截图。



图 11-7　密室逃生游戏

其实 Cordova 的文件系统也与密室逃生游戏非常类似，都是要通过某个“物品”来获取另一个“物品”，而靠着新获得的这个“物品”又能获得新的“物品”，这样依次循环下去就像解谜一样最终才能达到目的（逃出密室或者是实现对文件的某种操作）。

在密室逃生游戏中可能出现的东西有很多，比如钥匙、盒子、水果刀甚至是一卷卫生纸等，而在 Cordova 的文件系统中，用到的东西就比较简单，只能是某个类或者对象。



笔者就先做一个假设，因为 Cordova 中的类和对象的总数要比密室逃生中东西的种类少很多，因此学好 Cordova 中的文件操作比密室逃生要简单许多。

笔者列出 Cordova 中与文件操作相关的所有类，并将它们分为“介绍过的”和“没介绍过的”两组。

在之前介绍过的类有：

- FileReader
- FileWriter
- FileSystem
- FileTransfer
- FileEntry
- DirectoryEntry
- DirectoryReader
- FileUploadOptions
- File
- FileTransfrtOptions

下面这些类在之前没有具体介绍过：

- FileError
- FileTransferError
- FileUploadResult
- Flags
- LocalFileSystem
- Metadata

笔者将先对这些之前没有介绍过的类进行简单的介绍。

FileError 对象用来记录 Cordova 文件系统操作的一些错误信息。在 Cordova 中，任何一个文件操作发生错误时，都会在回调函数中返回一个 FileError 类的对象，用以告知开发者究竟是发生了什么错误导致了操作的失败。主要错误原因如表 11-9 所示。

表 11-9　FileError 可以表达的错误类型

错误名称	说明
NOT_FOUND_ERR	没有找到相应的文件或者目录
AABORT_ERR	终止操作发生错误
NOT_READABLE_ERR	文件不可读

（续表）

错误名称	说明
ENCODING_ERR	编码错误
NOT_MODIFICATION_ALLOWED_ERR	写入文件被拒绝
INVALID_STATE_ERR	无效的文件状态
SYNTAX_ERR	语法错误
INVALID_MODIFICATION_ERR	非法的修改请求
QUOTA_EXCEEDED_ERR	系统资源不足
TYPE_MISMATCH_ERR	类型匹配错误
PATH_EXISTS_ERR	创建已经存在的路径
SECURITY_ERR	其他错误

FileTransferError 类与 FileError 类类似，当文件上传发生错误时会返回一个 FileTransferError 类的对象给回调函数，它所能表达的错误类型比较少，只有三种，如表 11-10 所示。

表 11-10　FileTransferError 可以表达的错误类型

错误名称	说明
FILE_NOT_FOUND_ERR	文件未找到
INVALID_URL_ERR	无效的 URL
CONNECTION_ERR	连接错误



之所以 FileTransferError 类比 FileError 类所能够表达的错误类型在数量上要少很多，主要是由于 FileError 对所有的文件操作都产生效果，而 FileTransferError 仅针对文件上传有效。

FileUploadResult 的作用与 FileTransferError 又非常类似，只不过它发生作用是在上传文件成功时返回上传消息，它也包含了三个参数，如表 11-11 所示。

表 11-11　FileUploadResult 所附带的参数

参数名称	说明
bytesSent	上传文件所发送的总字节数
responseCode	HTTP 响应代码
response	HTTP 响应

类 Flags 和 Metadata 的内容其实在前面的章节已经提到过，Flags 作为 DirectoryEntry 对象的 getFile()和 getDirectory()方法的参数使用，包含了 create 和 exclusive 两组属性，用来指示当要获取的文件在文件不存在时是否由系统创建一个新的文件。

按照官方文档的说法，Metadata 代表一个文件或者是目录的状态信息，但是就目前来看这些状态信息仅有 modificationTime（用来记录文件或目录的最后修改时间）这一个属性，也许在今后 Cordova 的开发团队会为它加入新的内容，但是就目前来说这个类显得有些鸡肋。



目前来说 Cordova 中有不少类或者是方法确实由于缺乏实用性而显得它在设计上貌似不大合理，但是通过这种“糟糕的”设计还是能发现实际上 Cordova 预留了不少供将来扩展的缺口。因此，我们有理由相信 Cordova 最终会是一款比目前更加强大和好用的开发框架。

最后再说说 LocalFileSystem 类，这个类在之前的范例中已经被使用过（范例 11-2 的第 12 行）。该类可以用来定义所申请的文件系统权限是否为“持久化的”（在范例 11-2 中就是此种用法）。同时它还包含了两种方法 requestFileSystem（用来获取一个 FileSystem 类对象）和 resolveLocalFileSystemURI（通过本地 URI 参数检索 DirectoryEntry 或 FileEntry）。

下面再回到本章开头的范例 11-1，虽然当时笔者已经对代码中的一些内容进行了介绍，但是在本节中笔者将再以游戏“密室逃生”中的一些观点来带领读者重新分析一下它的内容。

为了方便读者阅读，现选出范例 11-1 中部分代码如下：

```
01      function onDeviceReady() {
02        window.requestFileSystem(LocalFileSystem.PERSISTENT, 0, gotFS, fail);
03      }
04       // 读取文件 readme.txt
05      function gotFS(fileSystem) {
06          fileSystem.root.getFile("readme.txt", null, gotFileEntry, fail);
07      }
08       // 获取文件句柄
09      function gotFileEntry(fileEntry) {
10            // 读取文件句柄
11          fileEntry.file(gotFile, fail);
12      }
13       // 读取文件内容
14      function gotFile(file){
15          readDataUrl(file);
16          readAsText(file);
17      }
18       // 以 Base 64形式读取文件
19    function readDataUrl(file) {
20          var reader = new FileReader();
21          reader.onloadend = function(evt) {
22              // 下面两行用来以 Base 64形式显示读取的，文件的此处先注释掉，读者可自行查看
```

结果

```
23            // alert("Read as data URL");
24            // alert(evt.target.result);
25         };
26         reader.readAsDataURL(file);
27      }
28      // 以文本方式读取文件
29      function readAsText(file) {
30         var reader = new FileReader();
31         reader.onloadend = function(evt) {
32            alert("以文本形式读取");
33            alert(evt.target.result);
34         };
35         reader.readAsText(file);
36      }
37      // 读取文件失败
38      function fail(evt) {
39         alert(evt.target.error.code);
40      }
```

想象你现在被关在一个房间里，被告知在这个房间里藏着一本书，在这本书中会有让你逃出这个房间的咒语。这本书就是在范例中要被读取的文件（file，在第 19~27 行中），而文件的内容就是能够让你逃出房间的咒语。

可是，按照游戏的设定，这本书不是谁都能碰的，没有“神器”在身的人如果碰了这本书就会立刻化成血水（可能是前一段时间受各种版本《封神榜》的影响，结果比喻的这么恶心）。于是现在就需要找到一个“神器”来帮助我们接触到这本书。这个“神器”就是之前介绍过的 FileReader，它的作用就是将一个 file 类的对象变成“可读的”。比如范例第 29~36 行就是这样的作用。

是不是这样就能够逃出“密室”了呢？当然不是，因为不管是“神器”还是带有咒语的书都是虚无缥缈的传说，不是谁都能找到的。因此，为了能够找到它们，就必须要有某种线索。比如第 6 行申请了文件系统的权限；而第 9~12 行中获取文件句柄就好比是找到了冒险的地图，这样才能够最终找到书（file）。



接下来可以按照类似的思路对“写”文件的功能做一些分析来加强自己的理解。

11.8 小结

本章介绍了 Cordova 中的文件系统，通过它可以实现对手机中的文件和目录进行读、写

等操作。但是请读者注意，由于 JavaScript 脚本本身运行速度的局限性，还是尽量不要去用它来开发一些诸如“文件管理器”或者是“小说阅读器”这样资源占用比较大的功能。那么是不是说 Cordova 的文件处理功能就一无是处了呢？当然不是，Cordova 的文件处理功能在实现一些简单的操作（比如图片资源的上传、下载还有简单的读写，如在音乐播放器中读取 LRC 文件）时相对于原声 SDK，还是有很大的优势的。在实际开发时一定要扬长避短，选择最适合自己的方法。

第 12 章
多媒体资源的捕获

在本书的第 10 章中，曾经介绍过一个 Media 类，通过它可以实现对音频的录制和播放等操作。但是媒体文件有很多，并不止有音频一种。那么其他类型的媒体文件（比如图片、视频）又要通过何种方式来捕获呢？Cordova 专门提供 Capture 对象来实现系统对图像、音频和视频资源采集的调用。

本章的主要知识点有：

- Capture 对象以及使用 Capture 对象进行操作时会调用的各种参数。
- 使用 captureAudio()方法采集音频并上传到服务器。
- 使用 captureImage()方法采集图片文件并上传到服务器。
- 使用 captureVideo()方法采集视频文件并上传到服务器。

12.1 声音的采集

Capture 在英文中包含了俘虏、捕获等含义，在实际使用中也可以近似地被翻译为采集。由此就可以联想到，它没有 Media 类中已经学习过的播放、暂停等功能，它所能实现的只有录制或者摄制功能。那么它与 Media 类的录音功能又有什么区别呢？接下来将以一个使用 Capture 对象来进行声音采集的例子对它做出说明。

首先在命令行执行命令添加相应插件：cordova plugin add cordova-plugin-media-capture。

【范例 12-1】利用 Capture 对象实现音频的采集。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <title>利用 Capture 类实现音频的采集</title>
05  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
06  <script type="text/javascript" charset="utf-8" src="js/cordova.js">
</script>
07  <script type="text/javascript">
```

```
08      // 开始音频采集，该方法由 onclick 动作调用
09      function myptureAudio() {
10          // 执行录音采集操作并分别设置采集成功和失败的回调函数
11         navigator.device.capture.captureAudio(captureSuccess, captureError,
{limit:2});
12      }
13      // 采集成功
14      function captureSuccess(mediaFiles) {
15         var i, len;
16          // 采集功能可以同时录制多段音频，录制完成后自动进行上传
17         for (i = 0, len = mediaFiles.length; i < len; i += 1) {
18            uploadFile(mediaFiles[i]);
19         }
20      }
21      // 采集失败
22      function captureError(error) {
23         alert("采集失败");
24      }
25      // 上传采集到的文件
26      function uploadFile(mediaFile) {
27         var ft = new FileTransfer(),
28            path = mediaFile.fullPath,
29            name = mediaFile.name;
30          // 使用 upload()方法
31         ft.upload(path,
32            "http://202.118.89.139/index.php",
33            function(result) {
34               console.log("上传成功");
35            },
36            function(error) {
37               console.log("上传失败");
38            },
39            { fileName: name });
40      }
41  </script>
42  <style>
43  * { margin:auto;}
44  body { background:#999; }
45  #button
```

```
46 {
47     width:240px;
48     height:100px;
49     margin-top:80px;
50     background:#f00;
51     font-size:40px;
52     color:#fff;
53     line-height:100px;
54     text-align:center;
55 }
56 </style>
57 </head>
58 <body>
59    <div id="button" onclick="myptureAudio();">开始录音</div>
60 </body>
61 </html>
```

运行之后的界面如图 12-1 所示，点击屏幕上方的“开始录音”按钮，设备开始进行音频的采集，可以在 LogCat 的日志中观察音频被上传的情况，如图 12-2 所示。在观察到日志中提示“上传成功”之后也可以直接查看服务器中是否被上传了音频文件。



图 12-1　程序运行后的界面

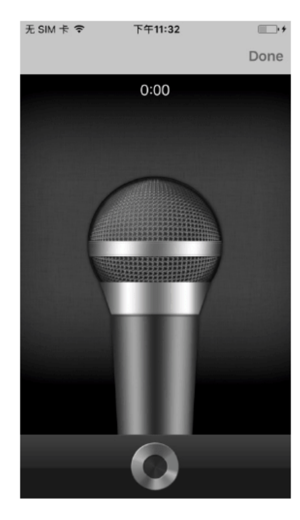


图 12-2　点击“开始录音”后的界面

相信读者已经得到了想要的结果，不过不知道读者有没有发现两个非常诡异的问题？

第一：在本例中并没有加入“停止录音”的功能，系统是怎么知道何时结束录音的，难道是作者忘记了？

第二：为什么会有两次“上传成功”的提示？难道这段范例里面加入了什么窃取个人隐私的代码吗？

当然不是这样的，不管之前有没有注意到这两个问题，通过对范例代码的讲解，读者很快就会明白这是为什么了。

首先，在页面中定义了一个按钮，并为它加入了点击事件 onclick="mypture Audio();"，如范例第 59 行所示。而在前面的 JavaScript 中，一切的操作也都是从自定义函数 myptureAudio()开始的。

在这个自定义函数中，只执行了一步操作：

```
navigator.device.capture.captureAudio(captureSuccess, captureError,
{limit:2});
```

它的作用是调用 capture 对象中的 captureAudio()方法开始对音频进行采集工作，其中 captureSuccess 和 captureError 分别是采集成功和失败时会执行的回调函数。可是除了这两个函数名以外，在 captureAudio()方法中还多出了一个参数{limit:2}。再联想到之前竟然将录音采集的结果上传了两次，有的读者就会猜想了，会不会是它的原因呢？

没错！真相就是这样的，正是该属性使得音频被上传了两次，可是为什么要上传两次呢？答案只有一个，那就是一共采集了两段音频。现在将展开这个参数进行介绍。

Cordova 对该方法的定义如下：

```
navigator.device.capture.captureAudio(CaptureCB captureSuccess,

CaptureErrorCB captureError,
 [CaptureAudioOptions options] );
```

首先，在 capture 之前加入了一段内容 navigator.device，这说明 capture 是作为一个全局变量被封装在 navigator 中使用的，因此在使用时不需要单独对它进行定义。接下来两个回调函数 captureSuccess 和 captureError 分别在采集成功和失败时被调用，当它们被调用时会默认接收到类型为 CaptureCB 或 CaptureErrorCB 的对象，用以保存采集到的信息。而第三个参数是可选的，它是一个 CaptureAudioOptions 类型的对象，包含了对音频采集时一些参数的设置，它包含的三个属性如表 12-1 所示。

表 12-1　CaptureAudioOptions 类中的属性

属性名	说明
limit	采集音频数量的最大值，默认值为 1
duration	采集时间
mode	选定的音频格式，该属性在早期版本中使用，现已被取消

也就是说，范例中的第 11 行实际上可以改写成如下的样子：

```
options.limit = 2;
options.duration = 10;
navigator.device.capture.captureAudio(captureSuccess, captureError,
```

```
options);
```

但是要注意的是，安卓系统并不提供对 duration 属性的支持，因此 Cordova 将会根据系统提供一个默认的采集时间，然后开发者可以通过设置 limit 属性的值来变相地改变采集时间。iOS 系统则不提供对 limit 属性的支持，使得采集的文件数量仅能为 1。

对音频采集成功之后就进入到了函数 captureSuccess 之中，系统会自动将采集到的数据存入一组 mediaFiles 类型的对象，它封装了采集到的媒体文件的一系列属性，如表 12-2 所示。

表 12-2　mediaFiles 中的属性

属性名	说明
name	记录被采集信息文件的文件名（不包含路径）
fullpath	记录被采集信息文件的文件名（包含路径）
Type	文件的 MIME 类型
lastModifiedData	文件最后被修改的时间，即采集完成的时间
size	文件的大小

这时被作为参数传递到 captureSuccess 之中的并不是一个单一的对象而是一个数组，因为 limit 属性存在的原因，被采集记录下的文件不止有一个，因此需要有数组来保存它们。范例第 17~19 行就是通过遍历的方法来实现这些文件上传的。



在论坛上有网友曾经反映过不知道该如何使用采集来的数据，因为在官方文档中没有这方面的介绍。其实这个问题非常简单，只要为 HTML 对象引入被采集的资源就可以了，比如本节中介绍到的音频文件就可以直接通过<audio src="song.wav" controls="controls">方式来使用，而获取资源的方法则是采用 mediaFiles 中的 fullpath 来实现。

本节的最后来总结一下 Capture 中对音频的采集与 Media 对象中录制声音的区别。

首先，在 Media 对象中录制的音频仅能播放而无法像 Capture 中一样作为文件被保存下来，更别说上传了。其次，Media 对象录制的音频可以选择随时结束录制，而 Capture 则必须提前设定好录制的时间。比如要做微信这样可以将声音进行传递的应用，使用 Capture 对象会比较方便进行上传；而如果是简单的录音回放，比如学习英语纠正发音的应用，则更适合使用 Media 对象。

12.2 图像信息的采集

介绍完了音频的采集，本节将继续介绍一个用户都喜闻乐见的功能——图像的采集，也就是常说的拍照。除了实现将采集到的图像上传到服务器之外，本节的范例中还要展示如何

将采集到的图像显示出来，具体实现方法如范例 12-2 所示。

【范例 12-2】图像文件的采集及使用。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <title> 图像文件的采集及使用</title>
05  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
06  <script type="text/javascript" charset="utf-8" src="js/cordova.js">
</script>
07  <script type="text/javascript" charset="utf-8">
08      // 开始采集图像
09      function mycaptureImage() {
10         navigator.device.capture.captureImage(captureSuccess, captureError);
11      }
12      // 图像采集成功
13      function captureSuccess(mediaFiles) {
14         var i, len;
15         for (i = 0, len = mediaFiles.length; i < len; i += 1) {
16            // 将采集到的图像显示出来

17               document.getElementById("photo_show").src=mediaFiles[i].fullpath;
18               // 上传图像文件
19                uploadFile(mediaFiles[i]);
20         }
21      }
22      // 采集图像不成功
23      function captureError(error) {
24         alert("采集失败");
25      }
26      // 上传采集到的文件
27      function uploadFile(mediaFile) {
28         var ft = new FileTransfer(),
29             path = mediaFile.fullPath,
30             name = mediaFile.name;
31           // 此处要自己搭建服务器
32         ft.upload(path,
33             "http://202.118.89.139/index.php",
34             function(result) {
```

```
35                  console.log("上传成功");
36              },
37              function(error) {
38                  console.log("上传失败");
39              },
40              { fileName: name });
41      }
42  </script>
43  <style>
44  * { margin:auto; background:#999; }
45  #photo
46  {
47      width:400px;
48      height:400px;
49      background:#f00;
50      margin-top:30px;
51  }
52  #button
53  {
54      width:400px;
55      height:100px;
56      background:#f00;
57      margin-top:30px;
58      color:#fff;
59      font-size:48px;
60      text-align:center;
61      line-height:100px;
62  }
63  </style>
64  </head>
65  <body>
66      <div id="photo">
67          <!--将图像在这里显示出来-->
68          <img width="100%" height="100%" src="img/logo.png" id="photo_show"
/>
69      </div>
70      <div id="button" onclick="mycaptureImage();">采集</div>
71  </body>
72  </html>
```

运行后的界面如图 12-3 所示，其中上方 Cordova 的 Logo 是打包在 asset/www 目录下的图片 Cordova.jpg。点击屏幕中下方的“采集”按钮，系统将先跳转到拍照界面利用摄像头采集图像信息并上传，最终显示在原本 Cordova Logo 的位置，如图 12-4 所示。



图 12-3　范例运行后的界面



图 12-4　采集到的图像在屏幕上被显示出来



在进行此范例的调试时，为了方便对摄像头的调用可以采用真机测试。

该范例所使用的方法与范例 12-1 极其的类似，因此这里不再对范例的内容做过多的讲解。唯一新加入的内容是，在第 17 行中将获得的图片路径加入到了 HTML 页面的 img 标签中，与在 JavaScript 中的使用方法几乎一致。与之类似的是在下一节中将要采集的视频文件，也可以用类似的方式展示在页面上。

另外，在实际操作中，点击“采集”按钮之后，手机将会先跳转到拍照的页面，点下拍照后才会再跳回图 12-4 所示的界面，包括在下一节中要介绍的视频采集也有类似的缺陷。

相信读者都玩过人人网或者是 QQ 空间。手机版的人人网和 QQ 空间都有一个“拍照上传”的功能就可以这样实现。

在对声音进行采集时，使用了一个用来设置采集参数的 CaptureAudioOptions 对象，在实现对图片的采集功能时同样也会用到一个类似的参数，那就是 CaptureImageOptions，它的属性如表 12-3 所示。

表 12-3　CaptureImageOptions 中的属性

属性名称	说明
limit	采集图片的数量
mode	采集图片保存的格式

12.3 视频的采集

介绍完了 Cordova 对音频和图像文件的采集后，本节就来介绍 Capture 的最后一个功能，即对视频的采集。范例 12-3 是采集视频并将视频发送到服务器的一个例子。

【范例 12-3】Capture 对视频的采集。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <title> Capture 对视频的采集</title>
05  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
06  <script type="text/javascript" charset="utf-8" src="js/cordova.js">
</script>
07  <script type="text/javascript" charset="utf-8">
08      // 采集视频
09      function mycaptureVideo() {
10          // 省略了可选参数 CaptureVideoOptions
11         navigator.device.capture.captureVideo(captureSuccess, captureError);
12     }
13      // 视频采集成功
14     function captureSuccess(mediaFiles) {
15         var i, len;
16         for (i = 0, len = mediaFiles.length; i < len; i += 1) {
17             uploadFile(mediaFiles[i]);
18         }
19     }
20      // 视频采集失败
21     function captureError(error) {
22          // 采集失败则弹出对话框
23         alert("采集失败");
24     }
25     // 上传采集到的文件
26     function uploadFile(mediaFile) {
27         var ft = new FileTransfer(),
28             path = mediaFile.fullPath,
29             name = mediaFile.name;
30     // 上传操作
31         ft.upload(path,
32             "http://202.118.89.139/index.php",,
33             function(result) {
34                 console.log("上传成功");
35             },
36             function(error) {
37                 console.log("上传失败");
38             },
39             { fileName: name });
40     }
41  </script>
```

```
42 <style>
43 * { margin:auto; background:#999; }
44 #button
45 {
46     width:400px;
47     height:100px;
48     background:#f00;
49     margin-top:30px;
50     color:#fff;
51     font-size:48px;
52     text-align:center;
53     line-height:100px;
54 }
55 </style>
56 </head>
57 <body>
58     <div id="button" onclick="mycaptureVideo();">采集</div>
59 </body>
60 </html>
```

运行后结果如图 12-5 所示，点击屏幕顶部的“采集”按钮可以跳转到录像界面开始视频的采集，之后可以在 LogCat 中看到“上传成功”的提示。



图 12-5 范例运行后的界面

与图像采集功能类似，该功能在进行一些传统摄像应用的开发时会显得非常无力。但是在进行一些社交应用的开发时，比如类似微信对讲功能的视频对讲器或者上传视频等，也还是不错的。

范例第 11 行使用参数 CaptureVideoOptions 对视频的采集进行设置，它的各项属性与 CaptureAudioOptions 相同，因此可参考表 12-1 的内容。

12.4 鸡肋的 MediaFileData 对象

在 Cordova 的采集功能中还有一个对象是 MediaFileData，它保存了被采集到的媒体文件所保存的数据，主要包含了 codecs（音频或视频文件的格式）、bittrate（音频或视频文件的比特率）、height（图像或视频的高度，单位为像素）、width（图像或视频的宽度，单位为像素）、duration（音频或视频的长度）。

这个对象貌似非常有用，但是这实在不是一个可靠的对象。表 12-4 中是安卓和 iOS 对其属性的支持情况。

表 12-4　安卓和 iOS 对 MediaFileData 对象中属性的支持情况

属性名	安卓	iOS
codes	不支持	不支持
bitrate	不支持	不支持
height	支持但存在问题	支持
width	支持但存在问题	支持
duration	支持但存在问题	支持

另外它的各个属性在黑莓系统中是完完全全地不支持。所以说 MediaFileData 对象是一个非常不靠谱的对象，在实际开发中要避免使用它，或者在使用时做好容错处理。



虽然说 Cordova 一直标榜能做到一次部署 7 大平台，且完全兼容，可是实际上还有许多细节不够完备，在兼容性无法得到本质性改变的前提下，就需要开发者发扬不怕失败的精神去多次尝试，找到最好的方案来避免这些问题。比如针对不同的平台进行微小的改动或者是避免去使用一些不完善的 API。

12.5 小结

本章介绍了 Cordova 中 Capture 对象的使用方法，由于它本身是一个全局对象，因此在学习时会比较容易上手，不过它并不具备十分完善的用户体验和兼容性，这也导致了想要用好它并不是一件非常容易的事情。但是总体来说，只要能够因地制宜地设计开发方案，Capture 对象还是能够带来不少惊喜的。

第 13 章

◂ Cordova本地存储的使用 ▸

对本地存储功能的支持是 HTML 5 的一大进步，同样地，在 Cordova 中也没有忽略这一功能。在 Cordova 中，Storage 类基于 W3C Web SQL Database Specification 提供了设备对本地存储的访问。在许多设备中，这些功能都是使用 HTML 5 的本地存储功能来实现的，只有少数设备由于不能很好地支持 HTML 5 的一些属性，而不得不靠一些底层的功能来实现。

本章的主要内容有：

- HTML 5 以及 Cordova 中本地存储系统的概念以及使用。
- 在 Cordova 中如何打开本地存储功能。
- 在 Cordova 中利用 SQL 语言实现对本地存储功能的调用。
- Cordova 本地存储中键值对的配对方法。

13.1 HTML 5 中的本地存储功能

作为开发者必须要有一个意识，那就是在设备上看到的内容实际上都是以某种形式被下载下来的，即使无法指出它被存储在哪个位置。比如每天浏览的网页，即使是在重复浏览昨晚刚刚看过的页面，而且内容与之前完全相同，也无法避免重新将该页面的内容“下载”一遍的命运。

但这并不意味着所有信息都需要被重复下载，比如每天要使用的微信或者 QQ，会将一些消息保存在本地而仅仅更新一些新增加的内容。比如美女刚刚发来的一张笑脸，这大概就是本地存储在日常生活中最被人们所熟知的一个应用了。那么本地存储究竟是什么呢？本章将抛开 Cordova 而先对本地存储做一个简单的介绍。

13.1.1 为什么需要本地存储

在 2011 年，笔者曾经为某大学开发过一款分享校园新闻的手机应用。由于当时移动开发还没有现在这么普及，因此虽然当时做出的样品非常粗糙还是得到了校方的高度肯定，我也为此洋洋自得了很长时间。

但是不久之后，悲剧发生了。有一天，该学校的负责人说由于笔者当时仅仅是对校园网

上的信息进行了简单的抓取，学生们每次想看通知都得将全部的通知都下载一遍，非常麻烦而且浪费流量。因此，要求我对原本的应用进行修改。

由此笔者想到了一个办法，就是在程序中加入一个标志变量，用来存储最新的通知 id，并将已经下载好的通知保存在本地。这样每次查询通知时就只需要更新新发布的通知了。我的这一做法再次得到了校方的肯定，一场危机就这么化解了。

后来笔者就想到了日常所浏览到的网页，不管是不是已经浏览过的内容，每次再去浏览时就需要重新下载，这就造成了很大的浪费。有没有办法来避免这种浪费呢？

有人提出了使用 cookie 方法，此方法比较常见的应用就是在一些网站登录之后再次访问时能够保存用户的登录状态，甚至是记录下之前曾经访问过哪些网站。当然也有些网站会利用 cookie 来获取用户感兴趣的事务，并以此为依据来打广告。

当然，使用 cookie 来记录一些本地信息是非常不方便而且不安全的，那么有没有什么方法能够既安全又方便地实现本地存储功能呢？

13.1.2　HTML 5 的本地存储

虽然说本地存储功能是 HTML 5 新功能中不可缺少的一部分，但是遗憾的是，一般所说的 HTML 5 本地存储，即 Web Storage 规范，却并不是 HTML 5 规范中的一部分。原本属于 HTML 5 规范的它由于不为人知的原因被从 HTML 5 中分离了出来。

除了 Web Storage 规范，HTML 5 中的本地存储还有 Web SQL Database 和 IndexedDB 两种。其中 Web SQL Database 使开发者可以在 Web 中方便地直接使用数据库，这一点让许多开发者爱不释手。相比之下 IndexedDB 受到了更多的冷落，目前仅被 Firefox 支持的它，甚至被 Mozila 表示永远不会使用。



非常可惜的是，Web SQL Database 事实上是一个已经被废除了的标准，因为世界上根本不存在叫做 SQL 的标准化查询语言，人们平常所说的 SQL 需要区分成 MySQL SQL、MS SQL、SQLLite SQL 等才能够被识别出来。但是目前来看这三种本地存储的规范对于 Cordova 本地存储功能的实现都有着极大的借鉴意义。

为了更好地理解 HTML 5 的本地存储机制，范例 13-1 中提供了一种简单地利用本地存储来实现访问次数记录的方法。

【范例 13-1】利用 HTML 5 中的本地存储功能实现访问次数的记录。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <title>利用 HTML 5中的本地存储功能实现访问次数的记录</title>
05  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
06  <script type="text/javascript" charset="utf-8">
07      // 记录上一次本页被访问的时间
```

```
08      var lastdata = null;
09      // 页面被加载时执行
10      function storage() {
11          // 如果有访问记录
12          if (localStorage.pagecount) {
13              // 记录数加一
14              localStorage.pagecount=Number(localStorage.pagecount) +1;
15              // 获取上次访问本页面的时间
16              lastdata = localStorage.newvisit;
17              // 获取当前系统时间
18              localStorage.newvisit=new Date();
19          }
20          // 本页没有被访问过
21          else {
22              // 创建访问记录并将次数记为1
23              localStorage.pagecount=1;
24              // 获取当前系统时间
25              localStorage.newvisit=new Date();
26          }
27      }
28      function count() {
29          // 本页被访问次数大于一，则说明有上次访问的时间记录
30          if(localStorage.pagecount > 1) {
31              alert("你访问了本页面" +
32                      localStorage.pagecount +
33                      "次" +
34                      "您上一次访问的时间是" +
35                      lastdata);
36          }
37          else {
38              alert("您访问了本页面" + localStorage.pagecount + "次");
39          }
40      }
41  </script>
42  <style>
43  #button { width:200px; height:200px; background:#f00; }
44  </style>
45  </head>
46  <body onload="storage();">
```

```
47      <h1 id="welcome">欢迎访问</h1>
48      <div id="button" onclick="count();"></div>
49  </body>
50  </html>
```

将页面保存为 localStorage.html 并运行，用鼠标点击屏幕左上角的红色方块，即可弹出对话框显示该页面被访问的次数，以及之前访问该页面的时间信息，如图 13-1 所示。



图 13-1　之前对该页面的访问被记录了下来

在范例中使用了 HTML 5 中的 localStorage()方法，在 HTML 5 标准中提供了两种可以实现本地存储的方法 localStorage 和 sessionStorage。其中 localStorage 的存储是永久性的，而 sessionStorage 存储的内容则会在浏览器关闭后被全部清除。在具体用法上它们两个没有太大区别，只需要为 localStorage 或者 sessionStorage 加入属性即可，如范例第 23 行所示。

在一些基于 HTML 5 的应用中就可以使用类似的方法在本地存储一些数据以避免重复下载，比如之前笔者做过的那款校园新闻应用，就完全可以采用 localStorage 方法来存储学生通知的内容，而不需要每次都从服务器上重复获取。此外，一些聊天记录，甚至是利用 Cordova 来做一款号码本或便签都可以用类似的方法来实现存储。

13.2 Cordova 中的本地存储功能

上一节介绍了 HTML 5 中的本地存储功能，那么在 Cordova 中的本地存储又是什么样的呢？与上一节介绍的内容又有什么联系？这些都是本节要解决的问题。

Cordova 的本地存储功能被设计成以键值对的形式来永久性地保存数据，比如以下代码：

```
Window.localStorage.setItem("keyname","被存储的内容");
```

当开发者想要使用被存储的数据时，就可以使用以下代码来调用：

```
var str = Window.localStorage.getItem("keyname");
```

而这种方法正是 IndexedDB 标准所推崇的。



有些读者可能不理解什么是键值对，举个例子吧。如果有一个表格，内容如图 13-2 所示，那么表格中每一项的列名与它唯一对应的内容就组成了一对键值对。比如周一对应的是馒头。

周一	周二	周三	周四	周五
馒头	米饭	大饼	拉面	面包

图 13-2　周一到周五的早饭构成的键值对

此外，Cordova 的存储 API 中还提供了一个简单的数据库，这显然是受到 Web SQL Database 的影响。虽然说目前没有一种通用的 SQL 语言，但是由于在移动设备中普遍预装的仅仅是一个 SQLLite，因此可以将 SQLLite SQL 的语法作为在 Cordova 中使用 SQL 语言的标准。

在实际操作中可以采用如下方法来新建一个数据库：

```
newDB = window.openDatabase(name,        // 数据库名称
 version,                                // 数据库的版本号
display_name,                            // 数据库的显示名称
 size                                    // 数据库初始大小
);
```

之后就可以像操作 SQL 一样对数据库进行操作，比如：

```
var sql_query = "SELECT * FROM tablename";               // 要执行的 SQL 语句
newDB.excuteSql(sql_query, onSuccess, onError);          // 执行
```

当然实际使用时比现在所介绍的内容要复杂得多，不过这并不影响它的便利性，甚至可以说 Cordova 中的本地存储功能是它最常用也是最好用的一组 API。



对此有疑议者，可以尝试在学完本章内容后分别使用 Cordova 的本地存储功能和文件系统功能来存储一组数据。

13.3 数据库的使用

在前面已经介绍过，开发者可以在 Cordova 中以 SQLite SQL 的语法来进行数据操作，在

进行这些操作时，就必须有一个数据库来提供这一切，这就是本节将要介绍的 Database 对象。

Database 对象为开发者提供了对数据库进行操作的可能，可以简单地理解为一个 Database 就是一个可以被操作的数据库的实体。它包含了两种方法 transaction 和 changeVersion，其中 transaction 很常用，一般通过它来执行一些语句，比如数据库的创建和数据的插入；changeVersion()方法一般不会被用到，它用来实现对数据库版本号的查询。

但是要怎样才能获得一个 Database 对象呢？可以使用 Storage 对象中的 openDatabase()方法实现。具体方法在上一节中已经介绍过，如下所示：

```
var newDatabase= window.openDatabase(name, version, display_name, size);
```



注意不是 createDatabae 而是 openDatabase，它的作用是创建并返回一个新的 Database 对象，不过当要被创建的数据库已经存在时，它还是会执行打开数据的操作。

在打开了数据库之后，自然要对它进行一些操作，Database 对象使用方法 transaction()来完成这些操作，具体方法如下：

```
newDatabase.transaction(populateDB,     // 要执行的操作
 error,                                 // 操作执行失败的回调函数
success);                               // 执行成功的回调函数
```

该方法中要执行的操作又牵扯到了一个新的对象，那就是 SQLTransaction 对象，它包含了允许用户对数据库进行操作的方法 executeSql()。通过 executeSql()方法，用户可以在设备上直接执行一条或者多条 SQL 语句。下面给出一个利用这个方法来对数据库进行操作的例子。

【范例 13-2】利用 Cordova 的 API 操作数据库。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <title>利用 Cordova 的 API 操作数据库</title>
05  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
06  <script type="text/javascript" charset="utf-8" src="js/cordova.js">
    </script>
07  <script type="text/javascript" charset="utf-8">
08      // 设置设备加载完毕的触发器函数 onDeviceReady()
09      document.addEventListener("deviceready", onDeviceReady, false);
10      // 设备加载完毕
11      function onDeviceReady() {
12          // 新建一个数据库，名为 myNewDatase
13          var myNewDatase = window.openDatabase("myNewDatase", "1.0", "a Test
```

```
Demo", 200000);
14              // 对数据库进行操作
15            myNewDatase.transaction(populateDB, error, success);
16        }
17        // 对数据库要进行的操作
18        function populateDB(tx) {
19             // 如果表 MYDEMO 已经存在则将它删除
20            tx.executeSql('DROP TABLE IF EXISTS MYDEMO');
21             // 创建一个名为 MYDEMO 的新表，其中包括两个字段 id 和 data
22            tx.executeSql('CREATE TABLE IF NOT EXISTS MYDEMO (id unique,
data)');
23             // 向表中插入一组数据
24            tx.executeSql('INSERT INTO MYDEMO (id, data) VALUES (1, "First
data")');
25             // 再插入一组数据
26            tx.executeSql('INSERT INTO MYDEMO (id, data) VALUES (2, "Second
data")');
27            // 再插入一组数据
28            tx.executeSql('INSERT INTO MYDEMO (id, data) VALUES (2, "Third
data")');
29        }
30
31        // 操作失败
32        function error(tx, err) {
33            alert("数据库操作失败"+err+tx);
34        }
35        // 操作成功
36        function success() {
37            alert("数据库操作成功");
38        }
39    </script>
40    </head>
41    <body>
42        <h1>利用 Cordova 的 API 操作数据库</h1>
43    </body>
44    </html>
```

编译之后运行结果如图 13-3 所示。



图 13-3 运行之后的结果发生了错误

没有想到，运行之后的范例竟然发生了错误，按道理来说，书中的例子应该全都正确才能显示出作者的技术精湛么？为此，作者表示，这个错误是为了引出另一个对象而特意留下的。这就是 SQLError 对象，它用来在数据库操作错误时记录错误的原因，关于它所包含的错误类型如表 13-1 所示。

表 13-1 SQLError 对象中记录的错误类型

错误类型	说明
UNKNOW_ERR	未知错误
DATABASE_ERR	数据库错误
VERSION_ERR	版本错误
TOO_LARGE_ERR	数据过大错误
QUOTA_ERR	超出数据配额错误
SYNTAX_ERR	语法错误
CONSTRAINT_ERR	约束错误
TIMEOUT_ERR	超时错误

本范例出现的错误类型被默认作为自定义函数 error 的参数传递了下来，如范例第 32 行所示。那么这到底是一种什么错误呢？

观察范例的第 22 行有一段 id unique 的内容，也就是说，新插入的内容中 id 的值必须是唯一的，而在范例的 26 行和 28 行中连续两次插入了 id 为 2 的内容，因此造成了数据插入的错误。将第 28 行删除后，运行结果如图 13-4 所示。

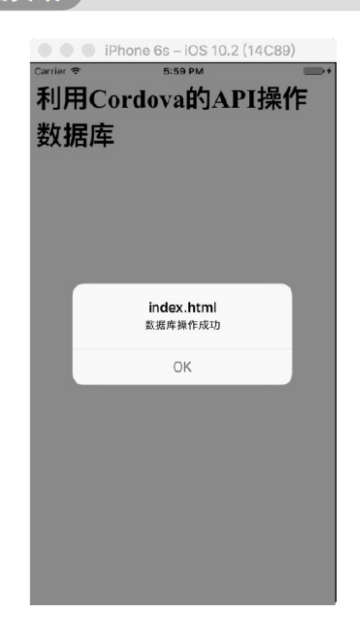


图 13-4　最终运行成功的结果



虽然说 SQLError 对象能够对出现的错误进行信息提示，但是一些本身操作上的错误确实无法识别，就好像汽车上的自动导航会在你偏离目标路线时给予提示，但当你设定了错误的目标后自动导航是无法察觉的。因此也就有了图 13-3 提示中的 undefined 这种错误类型。

介绍了这么多内容，其实还有一处重要的遗漏，那就是 SQL 语句究竟是怎样被执行的呢？通过范例的第 19~26 行也许能找到一些答案。

```
// 如果表 MYDEMO 已经存在则将它删除
tx.executeSql('DROP TABLE IF EXISTS MYDEMO');
// 创建一个名为 MYDEMO 的新的表，其中包括两个字段 id 和 data
tx.executeSql('CREATE TABLE IF NOT EXISTS MYDEMO (id unique, data)');
// 向表中插入一组数据
tx.executeSql('INSERT INTO MYDEMO (id, data) VALUES (1, "First data")');
// 再插入一组数据
tx.executeSql('INSERT INTO MYDEMO (id, data) VALUES (2, "Second data")');
```

可以看到 SQL 语句被包含在一个名为 executeSql()的方法中，这正是执行 SQL 语句时所使用的方法；但是前面那个 tx 又是什么呢？

其实这个对象在之前已经介绍过了，那就是 SQLTransaction，它被当作参数传递到了 populateDB 中，如范例第 18 行，这样就可以利用 SQL 语句来实现对数据库的操作了。

13.4 数据库内容的读取

上一节介绍了 Cordova 中对数据库进行操作的方法，但是不知道读者有没有一个疑问。数据库主要实现对数据的增、删、改、查等操作，如果增和删的功能都实现了，想必改的功能也是不难的，可是查询到的数据要怎样使用呢？这将是本节中要介绍的内容。

既然要从数据库中查询一些内容，就肯定要有相应的“容器”来存放它们，这个“容器”就是 SQLResultSet 对象。范例 13-3 展示了一种对数据库中内容进行查询的方法。

【范例 13-3】利用 SQLResultSet 对象实现数据库的查询。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <title>利用 SQLResultSet 对象实现数据库的查询</title>
05  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
06  <script type="text/javascript" charset="utf-8" src="js/cordova.js">
</script>
07  <script type="text/javascript" charset="utf-8">
08      // 设置设备加载完毕的触发器函数 onDeviceReady
09      document.addEventListener("deviceready", onDeviceReady, false);
10      // 向数据库中写入数据
11      function populatedb(tx) {
12          // 如果之前数据库已存在则将其删除
13          tx.executeSql('DROP TABLE IF EXISTS MYDEMO');
14          // 创建新的数据库对象，名为 MYDEMO
15         tx.executeSql('CREATE TABLE IF NOT EXISTS MYDEMO (id unique, data)');
16          // 插入一条数据
17          tx.executeSql('INSERT INTO MYDEMO (id, data) VALUES (1, "First
data")');
18          // 再插入一条数据
19          tx.executeSql('INSERT INTO MYDEMO (id, data) VALUES (2, "Second
data")');
20          // 再插入一条数据
21          tx.executeSql('INSERT INTO MYDEMO (id, data) VALUES (999, "so many
data")');
22      }
23      // 执行查询命令
24      function querydb(tx) {
25          // 查询表 MYDEMO 中的全部内容
26          tx.executeSql('SELECT * FROM MYDEMO', [], querySuccess, error);
27      }
28      // 查询成功
29      function querySuccess(tx, results) {
30          //记录读取的内容
31          var str = "";
```

```
32            // 数据库中数据的条数
33           var len = results.rows.length;
34           str = "共有: " + len +
35                   " 条记录,分别是" +
36                   "\n";
37            // 遍历从数据库中读出的记录
38           for (var i=0; i<len; i++){
39              str = str +
40                   "行数为 " + i +
41                   " ID : " + results.rows.item(i).id +
42                   " 内容  " + results.rows.item(i).data +
43                   "\n";
44           }
45            // 显示读出的内容
46            alert(str);
47       }
48        // 失败
49       function error(err) {
50           console.log("Error processing SQL: "+err.code);
51       }
52        // 成功
53       function success() {
54            // 打开数据库
55           var myNewDatase = window.openDatabase("Database", "1.0", "My Test
Demo", 200000);
56           myNewDatase.transaction(querydb, error);
57       }
58        // 设备加载完毕
59       function onDeviceReady() {
60            // 新建一个数据库
61           var myNewDatase = window.openDatabase("Database", "1.0", "My Test
Demo", 200000);
62            // 对数据库执行操作
63           myNewDatase.transaction(populatedb, error, success);
64       }
65   </script>
66   </head>
67   <body>
68       <h1>利用 SQLResultSet 对象实现数据库的查询</h1>
69   </body>
70   </html>
```

运行之后的结果如图 13-5 所示，可以看到在范例的第 17~21 行中被插入的数据已经全部被读取并显示了出来。



图 13-5　使用 SQLResultSet 对象读出了数据库中的内容

既然看到了结果，接下来就要看看这到底是怎么实现的？在范例的第 63 行开始执行对数据库的操作，而在它的回调函数 success 中却比范例 13-2 的同名函数中多加入了一些新内容（见第 55、56 行处）。首先使用 openDatabase()打开了名为 Database 的数据库，由于该数据在第 61 行中已经被创建（按照执行顺序应当先执行 61 行处的 openDatabase()方法），因此此处 openDatabase()方法的作用是打开一个已有的数据库，然后再对它进行操作（第 56 行）。

新的操作在自定义函数 querydb（第 24~27 行）中被执行，它所执行的只有一条 SQL 语句：

```
SELECT * FROM MYDEMO
```

即查询表 MYDEMO 中的全部内容。在该方法中仍然声明了两个回调函数，当 SQL 语句被执行成功时，将会进入函数 querySuccess 中，同时也会将一个类行为 SQLResultSet 的对象传入其中，它所包含的属性如表 13-2 所示。

表 13-2　SQLResultSet 类中的属性

属性名称	说明
insertId	数据库中记录的 id
rowAffected	SQL 语句执行后所作用到的记录的条数
rows	这是一个 SQLResultSetRowList 类型的对象，包含了读操作中获取的记录内容

范例第 33 行就利用了 SQLResultSet 对象中 rows 对象的 length 属性来返回数据库中记录的条数。既然已经提到了 rows，那么就有必要再介绍一下 SQLResultSetRowList 类，它的结构比较简单，除了已经使用过的 length 之外，就只有一个名为 item 的数组对象，其中包含了数据库中每条记录的值。如范例第 39~43 行就是对它的使用。



虽然本地数据库的使用大大地方便了开发者，但是由于 SQLLite 本身的局限性导致了本地存储功能不够强大，比如无法支持正则表达式、不能支持多表查询等。希望 Cordova 在今后的版本中能够加强这方面的支持。

13.5 键值对的使用方法

数据库是 Cordova 本地存储功能中最方便实用的一种 API，但是事无绝对，在某些特定情况下它就未必是最方便的了，因此就会有其他方法与它形成互补。这就是 Cordova 中的另一种存储方式，它的使用方法如范例 13-4 所示。

【范例 13-4】Cordova 本地存储中键值对的用法。

```
01  <!DOCTYPE html>
02  <html>
03  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
04  <head>
05  <script src="js/cordova.js" type="text/javascript" charset="utf-8">
</script>
06  <script type="text/javascript" charset="utf-8">
07      // 插入一组键值对
08      function charu() {
09          // 设置 key 对应的值为："本地存储"
10          window.localStorage.setItem("key", "本地存储");
11          // 获取 key 对应的键值，即"本地存储"
12          var value = window.localStorage.getItem("key");
13          // 将获取的内容显示在屏幕上
14          var para=document.createElement("p");
15          var node=document.createTextNode("插入了一条数据，现在 value 的值为：" +
value);
16          para.appendChild(node);
17          var scr=document.getElementById("screen");
18          scr.appendChild(para);
19      }
20      // 将本地存储中的键值对清空
21      function shanchu() {
22          // 执行清空操作
23          window.localStorage.clear();
24          // 再读取 key 对应的键值内容，存入变量 value 中
25          var value = window.localStorage.getItem("key");
26          // 显示 value 中的内容
27          var para=document.createElement("p");
28          var node=document.createTextNode("删除了一条数据，现在 value 的值为：" +
value);
29          para.appendChild(node);
30          var scr=document.getElementById("screen");
31          scr.appendChild(para);
32      }
33  </script>
34  <style>
35  /*屏幕上部的方块用于显示内容*/
36  #screen
37  {
38      width:90%;
```

```
39      height:300px;
40      margin:30px auto 0 auto;
41      border:1px solid #000;
42  }
43  /*按钮的样式*/
44  .button
45  {
46      width:90%;
47      height:60px;
48      line-height:60px;
49      font-size:40px;
50      font-weight:900;
51      color:#fff;
52      background:#f00;
53      margin:30px auto 0 auto;
54      text-align:center;
55  }
56  /*用于在顶部显示内容的字体样式*/
57  p
58  {
59      font-size:14px;
60      margin:0px 2% 0px 2%;
61      width:98%;
62      height:18px;
63      line-height:18px;
64  }
65  </style>
66  </head>
67  <body>
68      <div id="screen"></div>
69      <div class="button" onclick="charu();">写入数据</div>
70      <div class="button" onclick="shanchu();">删除数据</div>
71  </body>
72  </html>
```

运行之后的结果如图 13-6 所示。点击“写入数据”按钮，将会向 key 对应的键值对中写入内容“本地存储”，而当点击“删除数据”按钮时则会将本地存储中的全部键值对删除。



图 13-6　屏幕上部的显示部分能观察 key 对应键值对的内容

在本例中，只有第 10、12、23 和 25 行是与本地存储有关的内容，因此将针对这几处进行理解。第 10 行和第 12 行的作用是写入一对键值对并将它读取出来。在写入键值对的时候按照如下格式进行：

```
window.localStorage.setItem("键名", "键值内容");
```

当需要使用它时使用如下的方法进行调用（如范例第 12 行和第 25 行）：

```
var 变量名 = window.localStorage.getItem("键名");
```

在第 23 行中使用了 clear()方法，将全部键值对清除：

```
window.localStorage.clear();
```

通过点击“删除数据”按钮后的结果可以看出，当执行了该方法之后，key 对应的键值变成了 null，也就是空，说明确实是被清除了。

除了 clear 以外，其实还有一种用来删除键值内容的方法，这就是 removeItem()方法，用来实现对指定键值对的删除操作，使用方法如下：

```
window.localStorage.removeItem("键名");
```

最后一定还会有读者觉得使用键值对时，需要记录下每个键值对的键名非常不方便，所以说接下来还要介绍一种获取本地存储中键名的方法：

```
keyName = window.localStorage.key(0);
```

其中 keyName 中用来保存获取到的键名，而最后那个 0 则用来作为索引，以获取要使用被定义的第几组键值对的键名。

13.6 小结

本章从 HTML 5 的本地存储说起，介绍了 Cordova 中本地存储的使用方法，并对数据库式的查询和键值对的查询分别做了介绍。由于使用方法的不同导致了这两种方法有着不同的特点。比如要保存一系列的文章或电话号码，那肯定是数据库方式比较方便；但如果只是要保存一组数据，那么自然要首选键值对的方式。除此之外，还可以使用 HTML 5 本身的存储方式。本章虽然内容不多却需要读者不断地思考，在实际应用中选择最适合、最简便的方式来处理数据。



第三篇

项目实战篇

第 14 章 打造一款类Flappy Bird的小游戏

利用 HTML 5 容易学习和使用的特点，可以使开发者方便地完成一些小游戏，而 Cordova 则为这些小游戏最终的本地化提供了可能。相信在读者的心里，利用 HTML 5 来实现一款游戏还是有很大难度的，因此本章先不调用具体的 API，而是单纯使用 HTML 和 JavaScript 来实现一款简单的赛车游戏，使读者能够再一次体会到 Cordova 所带来的便利。本章将模仿年初非常流行的游戏 Flappy Bird 来对这一过程进行详细的描述。本章虽然不会提到对 Cordova 任何一处的具体调用，但是不得不说的是，利用原生的 HTML 5 来开发应用，再借助 Cordova 来打包的方式，本身就是对 Cordova 用法最本质的提炼。

本章主要知识点：

- 如何借鉴已有案例来实现自己的作品。
- 碰撞检测的原理和具体实现方法。
- 怎样将复杂的系统简化为简单、容易实现且能够被用户接受的系统模型。
- 怎样将复杂的元素简化，使之能够与自己的开发水平相适应。

14.1 需求分析

刚刚完成了 Cordova 基础知识学习的读者，有没有想做一点东西的冲动？在了解了 Cordova 是什么之后，有没有联想到可以不使用 Cordova 的 API，而仅仅利用它的打包功能来实现一款游戏？

想想小学刚开始写作文时的场景，几乎所有的小朋友都会仿照有限的几个模板进行模仿，于是就出现了“我和我的小伙伴们都惊呆了！”“我高兴得一蹦三尺高！”“我胸前的红领巾更鲜艳了！”这样流传已久的句子。

其实做一款游戏也是这样，最初可以从模仿开始，然后再做出一些小小的改动就算是自己的创作了。而当改动到了一定的程度就可以算是创新了。

几天前，笔者偶然发现身边的人在闲暇时都在玩着同一款游戏，如图 14-1 所示。在游戏中有一只被像素化了的“鸟”的图案，用户每点击一下屏幕它就会向上跃起，以此来躲避在它前进方向上的水管。如果不幸跃起高度太高或者太低，“像素鸟”就会宣告死亡，游戏结束。



图 14-1　热门游戏 Flappy Bird

当时笔者看到这款游戏的第一反应就是小时候玩过无数次的任天堂游戏“超级玛丽”的水下模式（如图 14-2 所示）。在游戏中“马里奥”可以通过“跳跃”来实现向上的动作，而当玩家不给它命令时，“马里奥”则会下坠。玩家可以通过“左右”和“跳跃”来操纵马里奥躲避水下的螃蟹乌龟等怪物，如图 14-2 所示。



图 14-2　“超级玛丽”水下版

这样分析起来就会发现虽然 Flappy Bird 是一款新游戏，而“超级玛丽”是 20 年前的老游戏，但是实际上 Flappy Bird 在制作难度上要比“超级玛丽”小了不少。首先“像素鸟”所需要躲避的钢管永远只向一个方向运动而且不存在类似海狗、海藻这样复杂的地形以及“金币”等元素，其次是 Flappy Bird 需要反应的仅有“跳跃”这个一维的动作，而不像“超级玛丽”那样还需要考虑多个方向。

这样看来，其实开发一款游戏并不是什么难事，尤其是在本章中，为了能让读者更直观地体验到这种化简的方法，笔者决定干脆把积分的功能也给省略掉，仅仅留下一个“像素鸟”不断跳跃的主体内容。

这样一来，本游戏需要实现的功能如下：

（1）“像素鸟”不断向屏幕的一侧飞行，当用户点击屏幕上的某个按钮时，它就会弹起，而当用户没有进行操作时它则会自由下落。

（2）在“像素鸟”飞行的过程中，会不断遇到各种长度的“钢管”，而为了将这一过程进一步简化，在本案例中仅保留了屏幕下方的钢管。

（3）当用户操纵的“像素鸟”与“钢管”发生碰撞时，“像素鸟”死亡，游戏结束。

由此，可以得到一个游戏运行时的大概示意，如图 14-3 所示。其中，红色的方块代表“像素鸟”，后面为蓝色的天空背景，绿色的长方形部分为原作中的“钢管”，而在本案例中笔者将把它替换成“树”的样子。

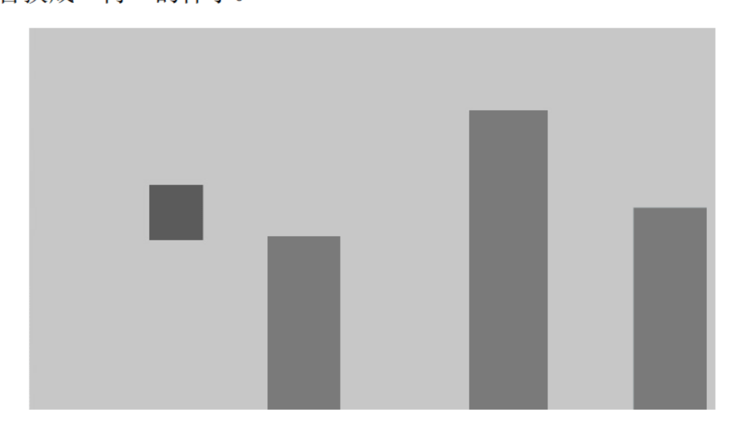


图 14-3　“山寨版”Flappy Bird 可能采用的布局

但是这个样子可能还会有读者认为太复杂了，那么干脆就对它进一步简化，使得屏幕中同时仅能显示一棵“树（长方形）”。这样一来，实际上需要处理的仅有“像素鸟”和“树”两个对象了。

这样就得到了最终的需求，在屏幕上显示一个“像素鸟”和一棵“树”。“像素鸟”在飞行的过程中可能会碰到迎面而来的大树，因此需要用户操纵它进行跳跃来对“树”进行躲避。

在下一节中，笔者还将对这个模型进一步简化，使之确实成为一款容易实现且具有一定可玩性的益智游戏。

14.2 模型建立

如果读者玩过“魂斗罗”一类的游戏，就会觉得实际上这类横版游戏在人物行走时还会遇到一个难点，那就是人物行动和地图行动的判断。

因为在横版游戏中，通常都是通过对背景的移动而不是人物的移动来实现人物移动的效果的，比如在如图 14-4 的人物画面中，当人物位于屏幕一侧时，通过人物移动而背景静止的方式来显示人物的运动。而当人物运动到屏幕中间位置时，人物的移动则是通过背景向左平

移来表现出人物向右运动的假象。

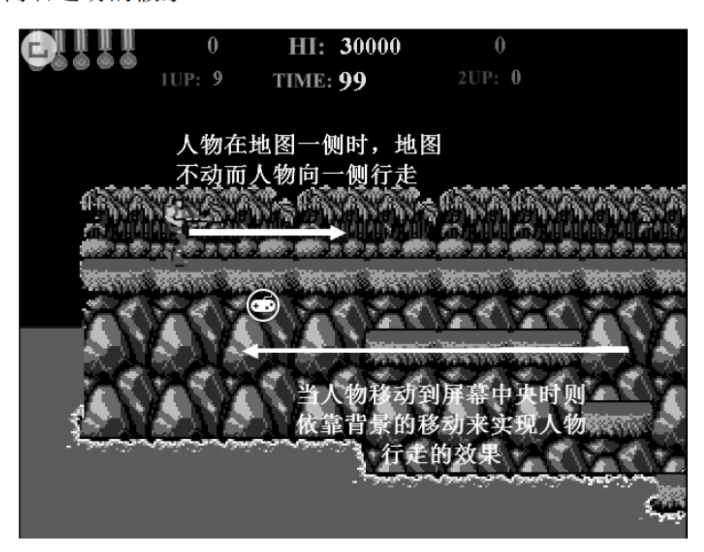


图 14-4　魂斗罗中的人物移动

而在本范例中，为了简化这一过程，决定将“像素鸟”“固定”在屏幕的中央位置，仅通过“树”向左的运动来表现“像素鸟”向右飞翔这一行为。因此在本范例中“像素鸟”实际上仅仅需要进行上、下动作的判断就可以了。

此外，本范例中虽然省略了游戏的积分部分，但是由于考虑到在本项目会用到一个作用于全局的计时器，因此可以简单地将游戏进行的时间作为分数显示在屏幕上。

由此，可以得出本游戏最终实现时的模型，如图 14-5 所示。

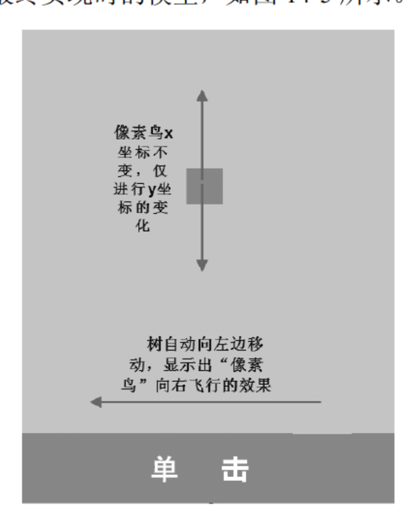


图 14-5　最终建立的模型

完成界面设计之后，将会针对这一模型设计相应的逻辑，以便最终能够实现这一模型。

14.3 界面设计

当模型设计完成之后，不应该急于去实现它的逻辑功能，而是要将游戏中各个要素先以一定的规律显示在屏幕上，这样不但有利于下一步逻辑设计的思路清晰，对于今后的测试和模型的进一步简化都有着非常大的帮助。

笔者实现了一个简单的界面，如范例 14-1 所示。

【范例 14-1】利用 HTML 和 CSS 实现游戏界面的布局。

```
01  <!DOCTYPE html>
02  <html>
03  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
04  <head>
05  <script src="js/cordova.js" type="text/javascript" charset="utf-8">
</script>
06  <script type="text/javascript" charset="utf-8">
07      // 这里是全局变量
08  </script>
09  <script type="text/javascript" charset="utf-8">
10      // 这里插入主要逻辑
11  </script>
12  <script type="text/javascript" charset="utf-8">
13      // 这里进行各个函数的设计
14      // 响应用户的点击操作
15      function jump() {
16          // 用户按下按钮
17      }
18  </script>
19  <style>
20  /*这里是页面中的样式，先以方块代替具体内容 */
21  * { margin:0px; }
22  #rec
23  {
24      width:400px;
25      height:609px;
26      position:absolute;
27  }
28  /*用来显示分数等信息*/
29  #num_rec
```

```
30  {
31      width:400px;
32      height:29px;
33      background:#000;
34      top:0px;
35      left:0px;
36      font-size:16px;
37      color:#fff;
38      line-height:29px;
39      text-align:left;
40  }
41  /*主要游戏区域*/
42  #game_rec
43  {
44      width:400px;
45      height:500px;
46      background:#007;
47      top:29px;
48      left:0px;
49      position:absolute;
50  }
51  /*按钮，点击后"像素鸟"跳跃*/
52  #button_rec
53  {
54      width:400px;
55      height:80px;
56      background:#9a3;
57      top:529px;
58      left:0px;
59      position:absolute;
60      font-size:48px;
61      font-weight:bold;
62      color:#fff;
63      line-height:80px;
64      text-align:center;
65  }
66  /*代表"像素鸟"的方块*/
67  #bird
68  {
```

```
69      width:50px;
70      height:50px;
71      left:175px;
72      bottom:200px;
73      background:#f00;
74      position:absolute;
75  }
76  /*代表大树的长方形*/
77  #tree
78  {
79      width:100px;
80      height:200px;
81      right:0px;
82      bottom:0px;
83      background:#0a0;
84      position:absolute;
85  }
86  </style>
87  </head>
88  <body>
89      <div id="rec">
90          <div id="num_rec">分数：0</div>
91          <div id="game_rec">
92              <div id="bird"></div>
93              <div id="tree"></div>
94          </div>
95          <div id="button_rec" onclick="jump();">跳跃</div>
96      </div>
97  </body>
98  </html>
```

运行之后的界面如图 14-6 所示。

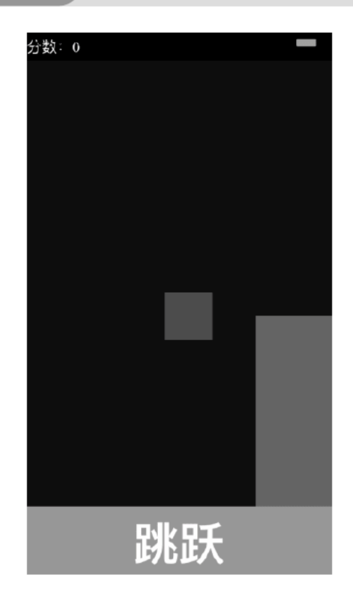


图 14-6　实现后的游戏界面



一般在使用 HTML 制作游戏时，主要采用绝对定位的方法来决定各个元素所处的位置。

有细心的读者可能注意到了，虽然各个元素都使用了绝对定位的方式来进行定位，但是它们的定位方式却是不尽相同的，比如“像素鸟”的定位（范例第 71 行和第 72 行）采用了 left 和 bottom 两个属性，而树木的定位（范例第 81 行和第 82 行）则采用了 bottom 和 right 两个属性。这又是为什么呢，把它们统一起来按照一般人的习惯全部使用 left 和 top 来进行定位难道不好吗？

统一起来自然是一种不错的方案，尤其是在使用现成的物理引擎时，能够有效地对游戏中的单位进行管理。但是本项目只是一个被简化了的例子，为了能够让读者更好地理解代码，它的每一个函数都是重新编写的。仅有的两个物体（像素鸟和树木）在进行碰撞检测时，使用的仅仅是像素鸟的底部坐标和树木的顶部坐标进行比较，因此使用 bottom 会比较方便。

在之前的设计中，希望屏幕上同时只存在一棵“树”，因此，使用 right 属性对树木进行判断，能够更加方便地确定“树”元素是否还保留在屏幕中。

至此，对界面的设计和实现就可以先告一段落了，下一节将着手实现 14.2 节所建立的模型。

在本范例的第 6~18 行，笔者已经为今后的逻辑实现部分分别预留了用来插入变量、流程判断，以及自定义函数的位置，在下一节中将向其中插入代码。

14.4 游戏的设计和实现

在前面的内容中，已经分析了这款小游戏所要实现的功能，并实现了游戏界面布局，本节将分成多个模块来实现游戏的各个功能。

14.4.1 “像素鸟”的飞行

说是“像素鸟”的飞行，其实倒不如说是实现树木的移动更为贴切一些。本节将设置一个计时器来实现树木向左不断移动的效果。具体实现方法如范例 14-2 所示。

【范例 14-2】实现树木的移动。

```
01  <!DOCTYPE html>
02  <html>
03  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
04  <head>
05  <script src="js/cordova.js" type="text/javascript" charset="utf-8">
</script>
06  <script type="text/javascript" charset="utf-8">
07      // 这里是全局变量
09      var tree_right = 0;                              //树的坐标
10      var tree_height = 200;                           //树的高度
11  </script>
12  <script type="text/javascript" charset="utf-8">
13      // 这里插入主要逻辑
14      var time_cul = setInterval("tree_move()",50);
15  </script>
16  <script type="text/javascript" charset="utf-8">
17      // 这里进行各个函数的设计
18      // 响应用户的点击操作
19      function jump() {
20          // 用户按下按钮
21      }
22      function tree_move() {                           // 使树移动
23          // 如果树已经移出页面，产生新树
24          if(tree_right > 400) {
25
26              tree_right = 0;                          // 对新树进行初始化
27
28              tree_height = Math.random()*360 + 60;       // 随机生成不同高度的树
```

```
29            }
30            else {
31                // 使树的 right 属性增加，树向左移动
32                tree_right = tree_right + 1;
33            }
34            set_tree();                        // 自定义函数，用来设置"树"显示的位置
35        }
36        // 设置树的位置
37        function set_tree() {
38            // 获取代表树的元素
39            var mytree = document.getElementById("tree");
40            // 设置树的位置
41            mytree.style.right = tree_right + "px";
42            // 设置树的高度
43            mytree.style.height = tree_height + "px";
44        }
45    </script>
46    <style>
47        /*此处省略若干行 CSS，可参考范例14-1的19~86行处 */
48    </style>
49    </head>
50    <body>
51        <div id="rec">
52            <div id="num_rec">分数：0</div>
53            <div id="game_rec">
54                <div id="bird"></div>
55                <div id="tree"></div>
56            </div>
57            <div id="button_rec" onclick="jump();">跳跃</div>
58        </div>
59    </body>
60    </html>
```

运行之后可以看到“树”会在屏幕中循环移动，并且每次能够刷新出不同的高度，如图14-7 所示。

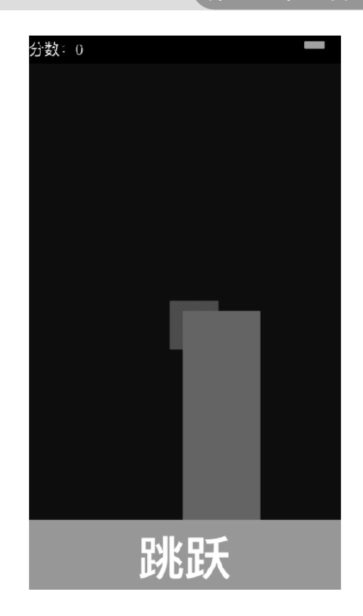


图 14-7 实现了“树”动功能

下面来介绍具体代码。首先需要明确的是，在利用 HTML 制作游戏时，几乎没有不使用函数 setInterval()的，因此在实现具体功能之前，首先就要习惯性地在程序中加入一段使用 setInterval()函数进行计时的代码，如范例第 14 行所示。



当然，一些棋牌类游戏（比如扫雷或者围棋等）可能是不需要计数器的，但是这类游戏不在我们讨论的范围之内。

接下来就要开始思考，“树”移动这一功能的本质是什么？毫无疑问，自然是树的横坐标改变了，在本范例中就体现为该元素 right 属性的变化。因此就可以先在被计数器调用的 tree_move 函数中加入一个新的自定义函数 set_tree。

那么就先实现 set_tree 函数的功能好了，即设置“树”元素的位置，为了方便，可以先设置一个全局变量 tree_right，如第 9 行所示。可是也不能光位置移动，如果是同一棵树反复出现游戏玩着会很无聊，于是就又加入了一个变量 tree_height 用于记录树木的高度。接下来就是用第 37~44 行将这些属性反映到屏幕上来。

既然能够让树显示了，那还得让它变化。没错，当初设置计数器就是这个目的，因此可以设置每次让“树”元素的 right 属性加 1，从而起到位置变化的作用。同时还要判断“树”是否超出了屏幕的范围，如范例第 26~35 行所示。

这样一来，整个“树”的移动功能就算是实现了，根据物体相对运动的法则，就可以理解为是“像素鸟”在向前飞行了。

14.4.2 “像素鸟”的跳跃和下落

以上介绍了使“树”移动的方法，本小节将继续介绍与移动有关的内容，只不过被移动

的对象变成了这款游戏的主角“像素鸟”。

【范例 14-3】实现“像素鸟”的跳跃和下落。

```
01  <!DOCTYPE html>
02  <html>
03  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
04  <head>
05  <script src="js/cordova.js" type="text/javascript" charset="utf-8">
</script>
06  <script type="text/javascript" charset="utf-8">
07      // 这里是全局变量
08      var tree_right = 0;
10      var tree_height = 200;
11      // 用于声明跳跃高度
12      var jump_height = 0;
13      var bird_bottom = 200;
14  </script>
15  <script type="text/javascript" charset="utf-8">
16      // 这里插入主要逻辑
17      var time_cul = setInterval("move()",30);
18  </script>
19  <script type="text/javascript" charset="utf-8">
20      // 这里进行各个函数的设计
21      // 响应用户的点击操作
22      function jump() {
23          // 用户按下按钮
24          jump_height = 5;
25      }
26      // 声明一个新的函数使"像素鸟"和"树"都可以运动
27      function move() {
28          tree_move();
29          bird_move();
30      }
31      // 此处省略了 tree_move()和 set_tree()两个自定义函数，可参考范例14-2的22~44行
32      // 使"像素鸟"移动
33      function bird_move() {
34          // 先判断是否飞得过高
35          if(bird_bottom > 450 ) {
36              // 飞得过高则不能再跳跃
```

```
37                jump_height = 0;
38            }
39            // 如果当前还在跳跃过程中
40            if(jump_height > 0 ) {
41                // 增加高度
42                bird_bottom = bird_bottom + 10;
43                jump_height = jump_height - 1;
44            }else{
45                // 降落
46                bird_bottom = bird_bottom - 1;
47            }
48            // 将结果显示出来
49            set_bird();
50        }
51        // 设置鸟"像素鸟"位置
52        function set_bird() {
53            // 获取代表树的元素
54            var mybird = document.getElementById("bird");
55            // 设置树的位置
56            mybird.style.bottom = bird_bottom + "px";
57        }
58    </script>
59    <style>
60    /*此处省略若干行 CSS，可参考范例14-1的19~86行处 */
61    </style>
62    </head>
63    <body>
64        <div id="rec">
65            <div id="num_rec">分数：0</div>
66            <div id="game_rec">
67                <div id="bird"></div>
68                <div id="tree"></div>
69            </div>
70            <div id="button_rec" onclick="jump();">跳跃</div>
71        </div>
72    </body>
73    </html>
```

运行之后的效果如图 14-8 所示。点击屏幕底部的“跳跃”按钮可以看到“像素鸟”猛地向上一冲，之后又缓缓地下降。本范例特意加入了一个边界检测的功能，也就是说“像素

鸟”绝对不会飞到屏幕中过高的地方。

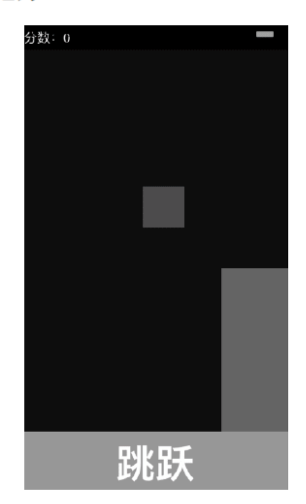


图 14-8 “像素鸟”跳跃到了高空中



想一想为什么笔者没有考虑“像素鸟”下落过低落到屏幕底部的情况呢？答案将会在下面给出。

其实读者自己阅读代码就会发现，控制“像素鸟”的方法与控制“树”移动的方法大致相同，都是通过设置一个公共变量的数值来实现对元素位置的改变。只不过在“像素鸟”的移动操作中还要考虑用户按键使它跳跃的情况。

阅读代码的第 42 行，可以看到有一个让元素的高度加 10 的操作，就是它让“像素鸟”一下跃起的。为了使“跳跃”成为一个连续的动作而不至于太突兀，这个值不能设得太大，但是可以连续进行几次同样的操作来增加“跳跃”的高度。所以在第 12 行设置了变量 jump_height 来专门存放这个操作的次数。

除此之外，在进行这些操作之前还要进行两个判断，一个是判断“像素鸟”有没有飞出屏幕的范围（范例 35~38 行），另一处就是判断当前飞行的状态是跳跃还是下落（范例第 40 行）。

还有一处非常重要的改动在范例的第 27 行，新加入了一个自定义函数 move。因为现在同时有两个元素需要移动，甚至在以后的内容中还要加入碰撞检测的功能。因此就需要一个统一的地方来管理它们。这样一来就使用了函数 move 将原本的 tree_move 和在本范例中新定义的 bird_move 全部包括在其中了。



实际上，在进行实际项目开发时要尽量避免这种滥用公共变量的情况发生。但是由于这只是一个非常小的项目而且本项目的目的主要是为了使读者理解一款游戏的开发过程，还请各位读者忽略这个错误吧。

14.4.3　碰撞检测功能

通过以上的学习，已经能够轻松地实现对“像素鸟”的操作了，但是现在这款游戏看起来反而非常奇怪了。因为原本这款游戏是让玩家去躲避“树”的，但是现在玩家操纵的“像素鸟”却可以毫无危险地“撞”到树上。这有点像十几年前一些非常山寨的游戏机上的象棋游戏，虽然可以模拟出棋盘让玩家相互对弈，却没有任何限定的规则。

而本小节中的内容就是要为这款游戏加入规则，对于这款游戏来说规则只有一个，那就是不要撞树。

【范例 14-4】为游戏加入碰撞检测。

```
01  <!DOCTYPE html>
02  <html>
03  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
04  <head>
05  <script src="js/cordova.js" type="text/javascript" charset="utf-8">
</script>
06  <script type="text/javascript" charset="utf-8">
07      // 这里是全局变量
08      var tree_right = 0;
09      var tree_height = 200;
10      // 用于声明跳跃高度
11      var jump_height = 0;
12      var bird_bottom = 200;
13      // 记录游戏运行时间，也作为分数显示
14      var num = 0;
15  </script>
16  <script type="text/javascript" charset="utf-8">
17      // 这里插入主要逻辑
18      var time_cul = setInterval("move()",30);
19  </script>
20  <script type="text/javascript" charset="utf-8">
21      // 这里进行各个函数的设计
22      // 响应用户的点击操作
23      function jump() {
24          // 用户按下按钮
25          jump_height = 5;
26      }
27      // 声明一个新的函数使"像素鸟"和"树"都可以运动
28      function move() {
29          tree_move();
30          bird_move();
31          //check();
```

```
32          // "像素鸟"撞到了地上
33          if( bird_bottom < 0 ) {
34              // 游戏结束
35              game_over();
36          }
37          // 检测"像素鸟"是否撞树
38          if( tree_right >75 && tree_right < 225 && bird_bottom <
tree_height ) {
39              // 游戏结束
40              game_over();
41          }
42          // 显示游戏分数
43          document.getElementById("num_rec").innerHTML="分数：" + num;
44          num = num + 1;
45      }
46      // 省略部分内容，可参考其他小节的内容
47      // 游戏结束
48      function game_over() {
49          // 停止计数器
50          clearInterval(time_cul);
51          // 显示游戏失败信息
52          alert("游戏失败");
53      }
54  </script>
55  <style>
56      /*此处省略若干行 CSS，可参考范例14-1的19~86行处 */
57  </style>
58  </head>
59  <body>
60      <div id="rec">
61          <div id="num_rec">分数：0</div>
62          <div id="game_rec">
63              <div id="bird"></div>
64              <div id="tree"></div>
65          </div>
66          <div id="button_rec" onclick="jump();">跳跃</div>
67      </div>
68  </body>
69  </html>
```

运行之后故意使“像素鸟”撞树，可以看到游戏弹出对话框提示：游戏失败，并结束，如图 14-9 所示。

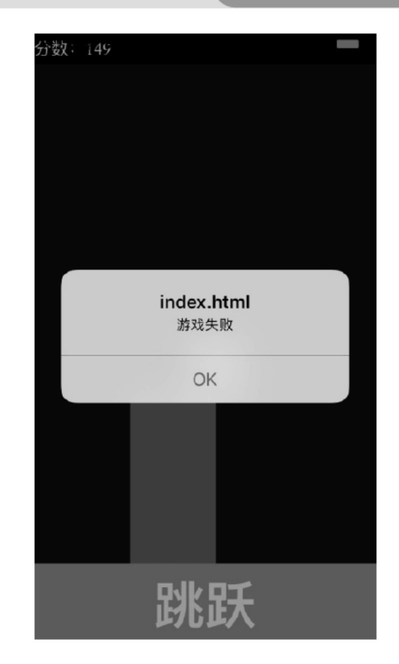


图 14-9　游戏碰撞检测的结果

按照最初的设定，无论是“像素鸟”撞到树上还是落到地上都算是游戏失败，游戏自动结束，见范例第 33 行和第 38 行的判断。

那么判断的数据又是怎么来的呢？笔者将利用图 14-10 所示来进行介绍。首先要明确几个尺寸的数值，即本范例中所使用的页面宽度是 400px，“像素鸟”的形状是一个长宽均为 50px 的正方形。“树”的宽度为 100px。那么在图 14-10 中给出两种临界位置，通过它的计算就可以得出，边界检测的上界和下界分别是 75 和 225 了。

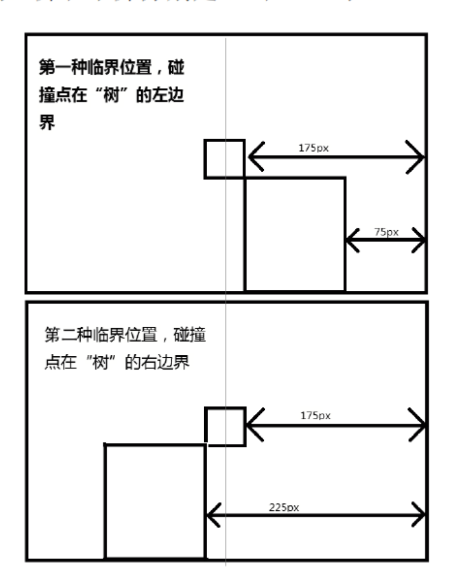


图 14-10　碰撞检测的两种边界条件

此外，本范例还加入了对游戏分数的显示，其实就是设了一个整数变量 num，当计数器每使用一次就会加 1，这样就可以将这个得到的数字直接作为分数显示出来了。



还记得前面那个问题吗——为什么不检测“像素鸟”落出下边界的情况呢？因为一旦它落到地上游戏就算失败了，因此不需要单独为它做一次检测。

14.5 界面的美化

以上已经实现了这款山寨 Flappy Bird 游戏的部分功能了，那么接下来要做的就是对当前丑陋的界面进行美化。首先要做的还是将那个方块换成一个可爱的“像素鸟”图案。

经过权衡，笔者决定使用游戏“超级玛丽”中的蘑菇头来作为“像素鸟”的替代者，如图 14-11 所示。将它保存为 bird.png 尺寸为 50×50px，然后就可以对页面中的 CSS 样式进行修改了。除了“像素鸟”图案之外，还要对天空背景加一个渐变的效果，而树则保持绿色不变。



图 14-11 最终被选中用来代替“像素鸟”的图案

具体的修改方案如下：

```
#bird
{
    width:50px;
    height:50px;
    left:175px;
    bottom:200px;
    /*此处为"像素鸟"加入蘑菇头图案，其他不变*/
    background:url(bird.png);
    position:absolute;
    z-index:1000;
}
#game_rec
{
    width:400px;
    height:500px;
    top:29px;
```

```
    left:0px;
    position:absolute;
    /*为天空加入渐变效果*/
    background-image:-webkit-linear-gradient(180deg,#34c,#abc);
    background-image:linear-gradient(180deg,#34c,#abc);
}
```

修改后的界面如图 14-12 所示。虽然还是有些不伦不类，但是要比修改之前好很多了。有兴趣的读者可以再试着为“树”、按钮等加入新的背景等元素，使它的界面变得真正好看起来。

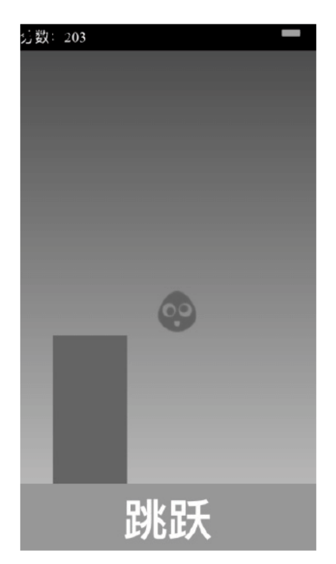


图 14-12　修改后的界面

完成之后就可以利用 Cordova 对它进行打包了，不过由于 Cordova 执行效率的问题，如果将范例中计数器的频率设置得过快，可能会导致游戏无法正常响应用户的点击操作；但是如果频率过低又会造成游戏进行缓慢，影响游戏的可玩性，这是一个非常大的遗憾。

14.6 缺陷和不足

经过了以上的修改，这款游戏终于能够出来见人了。但是它仍然存在着许多不足，而本节的内容就是要对这些不足之处进行分析。

14.6.1　玩法上的缺陷

在最初见到 Flappy Bird 这款游戏时，笔者还有些奇怪为什么要将钢管设置为上下两截呢？如图 14-13 所示。难道就像笔者在范例中设计的这样只有一截不好吗？

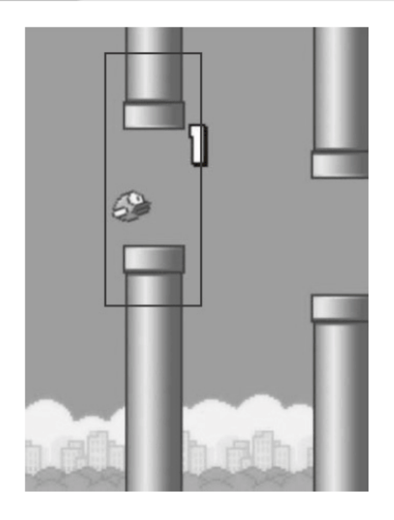


图 14-13　Flappy Bird 中的障碍设计

后来笔者就发现了一个有趣的问题。在本章所实现的这款游戏中，如果用户不停地点击底部的“跳跃”按钮，那么就永远不会出现“游戏失败”的结果，游戏会一直进行下去。而 Flappy Bird 中障碍物的这一设计就有效地避免了这一 bug。

那么，这个 bug 要怎么修改呢？笔者提供了两种方案可供选择：

（1）像 Flappy Bird 中一样，把“树”也设计成上下两截的。相信经过了本章的学习之后这不会构成什么困难。

（2）修改游戏的设定，当“像素鸟”（现在应该叫蘑菇头了）碰到屏幕顶部时同样显示游戏失败。

这两种方案修改起来都非常简单，不会给开发者带来太大的负担，因此读者有兴趣的话可以自行尝试一下。

14.6.2　功能上的贫乏

此功能依然要留给读者去完成，比如可以使用本地存储中键值对的方法来记录用户的最高分记录，或者是使用设备的震动效果来对用户的操作进行一些反馈等。

由于进行这些修改需要展示出太多无用的代码，以至于浪费许多篇幅，因此这些工作也只有靠读者去完成了。

14.6.3　人机交互不友好

应当再设计一个进入游戏的欢迎界面，甚至可以设计一个简单的说明来增强这款游戏容易上手的程度。此外还可以以小故事或者小漫画的形式为游戏加入一个简单的背景介绍。游戏“愤怒的小鸟”就是这方面一个非常不错的例子（见图 14-14 所示）。



图 14-14　“愤怒的小鸟”游戏中介绍背景的漫画

除此之外还应为游戏加入一些背景音乐等娱乐元素来增强用户的代入感，而这些功能都可以用前几章介绍过的 Cordova API 简单实现。

14.7 小结

本章实现了一个远远不够完善的小游戏，虽然它很简单而且有许多 bug，但是通过实现它的流程来学会设计游戏、实现游戏、完善一款游戏的思想是最重要的。此外，在学完了本章的内容之后，也要去思考有什么功能可以再加入到这款游戏中？而这些功能是使用了原生的 JavaScript 还是 Cordova 提供的 API？这样做有什么好处？笔者期待着有一天本书的读者也能做出像 Flappy Bird 这样的神作出来。

第 15 章

实战Cordova新闻应用

在 14 章的内容中，利用 JavaScript 实现了一个简单的 HTML 5 游戏，但是就目前来看，处理器的运算速度还无法支撑起足够的运算量以保证游戏的流畅运行。对于开发者来说，他们使用 Cordova 更希望能够实现一些信息管理类的应用，比如新闻浏览、在线商城等。这一类应用不但对资源的消耗较小，即使 Cordova 的运行效率总也不见长进，仍然能够担负起运行程序的重任。除此之外，这类应用还都比较容易上手，只要有一个简单的模板，开发者就能够轻松对这类应用进行大量复制，而这对目前“以量取胜”的 Cordova 开发者们来说尤其重要。

由于本章所要实战的项目是一个“巨大”的整体，因此就不得不涉及一些对服务端的操作，比如 PHP 后台的编写等，甚至要接触到一点“黑客”的内容。

本章的主要内容有：

- JSON 数据的格式以及如何获取和使用 JSON 数据。
- 使用 Ajax 获取其他页面上的内容。
- 如何实现“跨域”读取其他页面上的内容。

15.1 项目开始前的“闲言碎语”

对于这本书的读者来说，现在面临着一个重要的“问题”，那就是学习了 Cordova 之后到底能做什么。虽然本书的前几章对 Cordova 的某些特点（比如跨平台性、高效性等）进行了肯定性地介绍，但作为新世纪的程序员和创业者应该有辨别是非的能力。有没有考虑过现在的主流应用中为什么还没有见到过使用 Cordova 开发的程序。

事实上这样的应用是有的，只不过读者没有使用过或者使用过却没有意识到这是使用 Cordova 开发的。在这些应用中最有名的一款就是“豆瓣音乐人（如图 15-1 所示）”。笔者曾经使用过这款软件，得出的结果就是这实在不像是 Cordova 做出来的东西，一个重要的标志就是相对于笔者开发的某款音乐播放器来说，它流畅得实在是有些不像话。

后来笔者在网上找到了这款软件开发者的一篇日志，它揭示了他们是怎样使用 Cordova 达到近似原生应用效果的，而他们用来实现音乐播放功能的部分，自然引起了笔者最大限度地关注。从这里笔者了解到，原来他们并没有采用 Cordova 自带的 MP3 播放功能，而是使用

了 WebView 控件来实现音乐的播放。



图 15-1　豆瓣音乐人的界面

这与笔者一直以来对 Cordova 的一个观点不谋而合，虽然作为开发者都很热爱这种能够大大降低工作量的平台，但是从内心中并不能完全信任它。这一点主要表现在要尽量少地使用 Cordova 所提供的 API，而主要使用它能够将 HTML 文件打包成 APP 的功能。

根据“豆瓣音乐人”开发者的描述，在开发这款应用时，他们实际上仍然编写了大量原生代码来实现一些 Cordova 无法提供支持的功能，比如推送信息、状态栏提醒、绑定社交平台账号等。由此可见，“豆瓣音乐人”的开发者利用 Cordova 实现了一款不逊色于原生应用的应用。

但这并不是说普通开发者们就无法利用 Cordova 开发出好的应用了，如果是这样恐怕这本书就没有多大意义了。那怎样才能够在开发者还没有掌握非常高超的技术水平时，就能够利用 Cordova 开发出高水平的应用呢？那就是尽量只使用 HTML 5 中提供的功能，而避免使用 Cordova 提供的 API。

那么哪些功能的应用符合这样的需求呢？读者首先想到的大概是游戏，比如第 14 章利用简单的 HTML 和 JavaScript 实现了“像素鸟”的基本功能。可是实际上这游戏并不好玩，而且恐怕许多手机的处理器还不足以支撑起足够的运算量（运行像素鸟已经足够了，但是再复杂一些的游戏，如用 HTML 5 实现一款“愤怒的小鸟”就很难保证它在手机上不卡）。

其次就是一些新闻类应用了，比如对时事热点的浏览、类似“糗事百科”的条目阅读，甚至是某个领域商品信息的铺排。这些都是个人开发者们所热衷于开发的应用，能够为他们带来快速而且稳定的收入。最重要的是仅仅利用 JavaScript 就可以实现对网上信息的获取，这样就不必担心 Cordova 中 API 的种种不如意了。

还有非常重要的一点就是这类应用实现起来比较容易，一般开发者都能够很快地掌握，而且“换装”会非常简单。比如笔者开发了一款浏览大连新闻的应用，只需要将数据做一些修改就能够再得到一款浏览青岛新闻的应用，甚至改成浏览宇宙新闻都是很容易的。

因此，本章就利用 Cordova 实现一款浏览新闻的应用。

15.2 项目需求

本章所需要实现的功能非常简单，只需要在服务端生成新闻列表，然后在手机上利用 jQuery 的 Ajax 功能获取这些数据，最后将获取的数据显示出来即可。

此外，本项目的服务端将采用 PHP，因为 PHP 的资料还是最容易找到的，并且常用的 CMS（如织梦、DISCUZ、PHPCMS 等）均采用了 PHP，因此现阶段对于个人开发者来说，PHP 是最合适的。

15.3 界面设计和实现

首先还是来实现对界面的设计，本次需要的界面有两个，分别是首页的新闻列表和点开某个列表项之后显示的具体新闻内容。

15.3.1 新闻列表的设计和实现

由于本章所展示的仅仅是一个例子，因此笔者省略了一般此类应用常用的图片轮播等元素，但是必不可少的顶部栏仍然得到了保留。设计出的界面如图 15-2 所示。

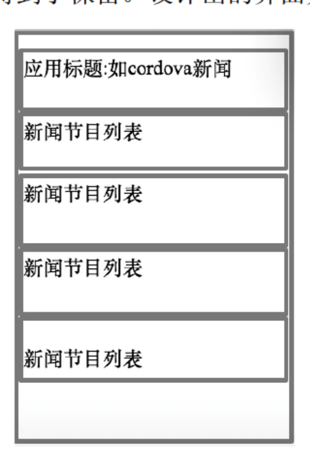


图 15-2　新闻列表界面布局

为了保证显示的效果，需要将顶部栏固定在屏幕的底部，而中央的新闻列表则可以根据用户的需要进行上下滑动。

具体实现方法如范例 15-1 所示。

【范例 15-1】新闻列表界面的实现。

```
01  <!DOCTYPE html>
02  <html>
```

```
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>新闻列表界面的实现</title>
06  </head>
07  <style type="text/css">
08  * { margin:0px; }                /* 清除页面自带边距 */
09  body { background:#0C9; }        /* 设置背景颜色 */
10  .main_rec                        /* 页面的全部内容均写在这个元素中 */
11  {
12      width:100%;
13      min-height:100px;
14      background-color:#0C9;
15      overflow:auto;
16  }
17  header                           /* 顶部栏 */
18  {
19      width:100%;
20      height:40px;
21      background-color:#006;
22      position:fixed;
23      top:0px;
24  }
25  header h1                        /* 顶部栏中的标题 */
26  {
27      width:100%;
28      height:40px;
29      text-align:center;
30      color:#FFF;
31      font-size:24px;
32      line-height:40px;
33      font-weight:bold;
34  }
35  .content                         /* 这里面显示具体新闻列表 */
36  {
37      width:100%;
38      min-height:60px;
39      margin-top:40px;
40      position:relative;
41      float:left;
```

```
42      overflow:inherit;
43  }
44  .list_rec                          /* 每一个列表项 */
45  {
46      width:96%;
47      height:36px;
48      margin:2px 2%;
49      background-color:#063;
50      float:left;
51  }
52  .list_title                        /* 列表项中的字体样式 */
53  {
54      width:98%;
55      height:36px;
56      float:left;
57      margin-left:2%;
58      color:#000;
59      text-align:left;
60      line-height:36px;
61      font-size:24px;
62  }
63  </style>
64  <body>
65      <div class="main_rec">
66          <header>
67              <h1>Cordova 新闻</h1>
68          </header>
69          <div class="content">
70              <div class="list_rec">
71                  <div class="list_title">新闻题目</div>
72              </div>
73              <!--此处省略部分内容，可将本范例第70~72行的部分复制多次进行补全 -->
74          </div>
75      </div>
76  </body>
77  </html>
```

运行后得到界面如图 15-3 所示。



图 15-3　实现后的新闻列表界面

在 header 元素中使用了属性 position:fixed 来保证顶部栏能够永远固定在屏幕顶部，如范例第 22 行所示。在具体的内容中为了保证其中的列表项不会溢出，就需要对它的 overflow 属性进行设置，如第 15 行所示。

除此之外，笔者将不再对本范例做过多介绍，请读者根据注释自行对范例中的内容进行理解。

15.3.2　新闻内容页的实现

与新闻列表的设计思想相同，具体的新闻内容界面也采取了尽量简单的设计思想，且在本范例中仅显示新闻的题目、作者以及具体内容。此外为了保持应用界面的一致性还要为新闻内容界面加入与新闻列表相同风格的顶部栏。为了保证用户在看完一条新闻之后能够顺利地返回新闻列表查看其他新闻，因此还要在头部栏中加入“返回”按钮，最终布局如图 15-4 所示。



关于“返回”按钮的位置一直是移动开发领域一个比较有争议的问题，因为按照一般人机交互的理论，应当将操作按钮尽量靠近屏幕的右下方位置以便用户进行单手操作。但是通过图 15-4 可以看到，放置“返回”按钮的位置恰恰是单手操作最难以触碰到的左上角。读者可以将其理解为是为了防止用户因为误触而有意为之，也可以理解为是习惯所致。

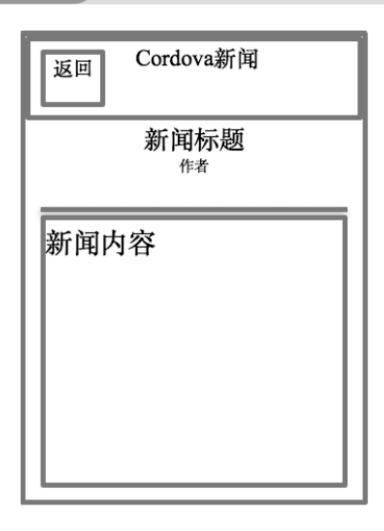


图 15-4　新闻内容界面布局

该界面的具体实现方法如范例 15-2 所示。

【范例 15-2】新闻内容界面的实现。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>新闻内容界面的实现</title>
06  </head>
07  <style type="text/css">
08  * { margin:0px; }                /* 清除页面自带边距 */
09  body { background:#0C9; }        /* 设置背景颜色 */
10  .main_rec                        /* 页面的全部内容均写在这个元素中 */
11  {
12      width:100%;
13      min-height:100px;
14      background-color:#0C9;
15      overflow:auto;
16  }
17  header                           /* 顶部栏 */
18  {
19      width:100%;
20      height:40px;
21      background-color:#006;
22      position:fixed;
```

```
23      top:0px;
24      z-index:999;                    /* 保证顶部栏不会被遮挡 */
25  }
26  header h1                           /* 顶部栏中的标题 */
27  {
28      width:100%;
29      height:40px;
30      text-align:center;
31      color:#FFF;
32      font-size:24px;
33      line-height:40px;
34      font-weight:bold;
35  }
36  #back                               /* 返回按钮 */
37  {
38      width:48px;
39      height:24px;
40      background-color:#f00;
41      position:fixed;
42      top:8px;
43      left:16px;
44      z-index:1000;
45      font-size:18px;
46      color:#FFF;
47      text-align:center;
48      line-height:24px;
49  }
50  .content                            /* 这里面显示具体新闻列表 */
51  {
52      width:100%;
53      min-height:60px;
54      margin-top:40px;                /* 保证不会被顶部栏遮挡 */
55      position:relative;
56      float:left;
57      overflow:inherit;
58  }
59  .content h1                         /* 新闻标题 */
60  {
61      width:100%;
```

```
62      height:60px;
63      text-align:center;
64      color:#000;
65      font-size:36px;
66      line-height:60px;
67      float:left;
68  }
69  .content h4                    /* 用来显示作者、日期等内容 */
70  {
71      width:100%;
72      height:25px;
73      color:#000;
74      font-size:12px;
75      text-align:center;
76      float:left;
77  }
78  #line                          /* 这是一条分割线 */
79  {
80      width:94%;
81      height:1px;
82      border:solid 1px #FFFFFF;
83      float:left;
84      margin-left:3%;
85  }
86  #neirong                       /* 新闻正文内容 */
87  {
88      width:92%;
89      float:left;
90      overflow:inherit;
91      margin:9px 3%;
92      color:#000;
93      font-size:16px;
94      line-height:24px;
95      text-indent:20px;          /* 文本的首行缩进 */
96  }
97  </style>
98  <body>
99      <div class="main_rec">
100         <header>
```

```
101                <div id="back">返回</div>
102                <h1>Cordova 新闻</h1>
103            </header>
104           <div class="content">
105                <h1>新闻题目</h1>
106                <h4>作者：猫爷  发表日期：2014-4-7</h4>
107              <div id="line"></div>
108              <div id="neirong">正文内容</div>
109           </div>
110       </div>
111 </body>
112 </html>
```

所实现的界面运行后如图 15-5 所示。

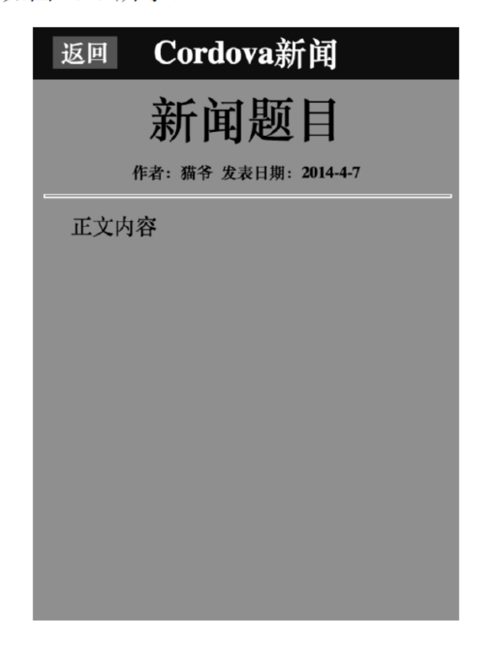


图 15-5　最终实现的新闻内容界面

读者可自行阅读代码进行理解，这里只介绍第 78~85 行处的 CSS 样式。因为有不少开发者在实现页面中一些分割线之类的效果时都会采用图片，这样对开发者来说可能会比较方便。可是为了保证页面的加载速度而且为今后的维护考虑，笔者建议页面中的元素还是尽量使用 CSS 来实现。



即使是彩色线条也是可以通过背景的渐变简单实现，这样虽然最初可能不大适应，但是长远来说比用图片要方便得多。

15.3.3 界面的进一步整合

前面的内容已经实现了新闻列表和新闻内容页的设计，不过现在还有一个问题。那就是由于本项目使用 Cordova 完成，因此在客户端只有 JavaScript 脚本可以使用，这就使得如何在两个页面间传递数据成为一个难题，虽然说 HTML 中提供了 window.location.search 的方法来实现在 HTML 间进行参数传递，可这毕竟还是不太方便，有没有能够替代它的方案呢？

回想本书中介绍过的知识，本地存储也许是个不错的想法，其实还有更好的办法。因为本项目中实际上只有两个页面，可以利用 JavaScript 将它们整合到一个页面中，不但能够有效提高页面间切换的速度，而且能够减少一些不必要的流量浪费。

【范例 15-3】将两页面整合为一个页面。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>将两页面整合为一个页面</title>
06  <script>
07  // 从新闻内容界面返回，显示新闻列表
08  function show_list() {
09      // 获取两个界面中的元素
10      var list = document.getElementById("list");
11      var news = document.getElementById("news");
12      // 对两界面中的元素进行显示和隐藏的设置
13      news.style.display = "none";
14      list.style.display = "block";
15  }
16  // 用户点击新闻列表项，跳至相对应的新闻内容页面
17  function show_news() {
18      // 获取两个界面中的元素
19      var list = document.getElementById("list");
20      var news = document.getElementById("news");
21      // 对两界面中的元素进行显示和隐藏的设置
22      list.style.display = "none";
23      news.style.display = "block";
24  }
25  </script>
26  </head>
27  <style type="text/css">
28  * {  }                              /* 清除页面自带边距 */
```

```
29  body {  }                             /* 设置背景颜色 */
30  .main_rec {  }                        /* 页面的全部内容均写在这个元素中 */
31  header {  }                           /* 顶部栏 */
32  header h1 {  }                        /* 顶部栏中的标题 */
33  #back {  }                            /* 返回按钮 */
34  .content {  }                         /* 这里面显示具体新闻列表 */
35  .content h1 {  }                      /* 新闻标题 */
36  .content h4 {  }                      /* 用来显示作者、日期等内容 */
37  #line {  }                            /* 这是一条分割线 */
38  #neirong {  }                         /* 新闻正文内容 */
39  .list_rec {  }                        /* 每一个列表项 */
40  .list_title {  }                      /* 列表项中的字体样式 */
41  </style>
42  <body>
43      <div class="main_rec" id="news">
44          <header>
45              <div id="back" onClick="show_list();">返回</div>
46              <h1>Cordova 新闻</h1>
47          </header>
48          <div class="content">
49              <h1>新闻题目</h1>
50              <h4>作者：猫爷  发表日期：2014-4-7</h4>
51              <div id="line"></div>
52              <div id="neirong">正文内容</div>
53          </div>
54      </div>
55      <!--上面部分是新闻内容，下面部分是新闻列表-->
56      <div class="main_rec" id="list" style="display:none">
57          <header>
58              <h1>Cordova 新闻</h1>
59          </header>
60          <div class="content">
61              <div class="list_rec" onClick="show_news();">
62                  <div class="list_title">新闻题目</div>
63              </div>
64             <!--由于篇幅问题，此部分省略了部分内容，读者可复制第61~63行内容进行补全-->
65          </div>
66      </div>
67  </body>
```

```
68  </html>
```

运行之后屏幕上显示首页新闻列表界面，用户随意点击一条新闻标题页面内容，就会切换到新闻内容界面，而当用户再点击左上角的“返回”按钮时，界面则会返回到新闻列表界面中去。

15.4 利用 Ajax 获取服务器上的信息

界面实现之后，要做的就是想办法将从网络上得到的信息显示在屏幕上了。这时就遇到了一个巨大的难题：怎样获取网络上的信息。这对 Web 开发者来说似乎非常容易，因为他们只需要选择一门脚本语言（ASP、PHP、JSP 甚至 Ruby）将要显示的内容上传到服务器上，用户就可以通过浏览器进行访问了。

但是对于 Cordova 开发者来说就变得非常困难了，排除掉难以获得信任的 API，可以使用的只剩下 HTML、CSS 和 JavaScript 了。准确点说，在实现获取来自网络的信息这一点上，能依靠的只剩下 JavaScript 了。

15.4.1 Ajax 的一个简单实例

好在 JavaScript 为开发者提供了一种可以与服务器进行异步交换数据并能够对页面进行更新的技术，叫做 Ajax。笔者毫不客气地说，不用异步，只要能更新数据就足够了。这就是范例 15-4 所实现的功能。

【范例 15-4】使用 Ajax 获取 JSON 数据。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>使用 Ajax 获取 JSON 数据</title>
06  <script src="http://code.jquery.com/jquery-1.11.1.min.js"></script>
07  </script>
08  <script>
09  function myAjax() {
10      // 使用 Ajax 的 getJSON 方法
11      $.getJSON("test.js",function(result){
12          // 遍历获取的 JSON 数据
13          $.each(result, function(i, field){
14              // 将获取的 JSON 数据显示出来
15              $("p").append(field + " ");
16          });
```

```
17      });
18  }
19  </script>
20  </head>
21  <body>
22      <p></p>
23      <h1 onclick="myAjax();">点击这里获取数据</h1>
24  </body>
25  </html>
```

此外，还需要在同一目录下保存内容{ "firstName": "Bill","lastName": "Gates","age": 60}到文件 test.js 中去。之后运行范例 15-4，点击屏幕上的文字，结果如图 15-6 所示（被框起来的部分是利用 Ajax 从 test.js 文件中读出的数据）。

Bill Gates 60 Bill Gates 60 Bill Gates 60

点击这里获取数据

图 15-6　利用 Ajax 获取其他文件中的 JSON 数据

接下来要做的就是将 test.js 上传到服务器，读者可以使用 Xampp、AppServ 等一键安装包在 Windows 下安装 Apache 以及 PHP 服务器环境。安装完成后可以在如图 15-7 所示的面板中启动服务器。

将 test.js 文件上传到 Apache 根目录之后，将范例 15-4 的第 11 行修改为：

```
$.getJSON("http://127.0.0.1/test.js",function(result)
```

然后再运行修改后的范例发现已经无法获取 test.js 中的 JSON 数据了，但是如果将范例代码也一同复制到 Apache 根目录下，通过浏览器执行范例确实能够获取 JSON 数据。这就说明了代码上是没有错误的，可是到底是哪里出了错误呢？

问题出在了 jQuery 的 getJSON 方法上，虽然可以通过 Ajax 访问其他文件或者页面上的数据，但是出于安全考虑却要求这两个页面必须处在同一个“域”中，这下就麻烦了，好不容易想到了获取数据的方法却不能使用，这该如何是好啊。

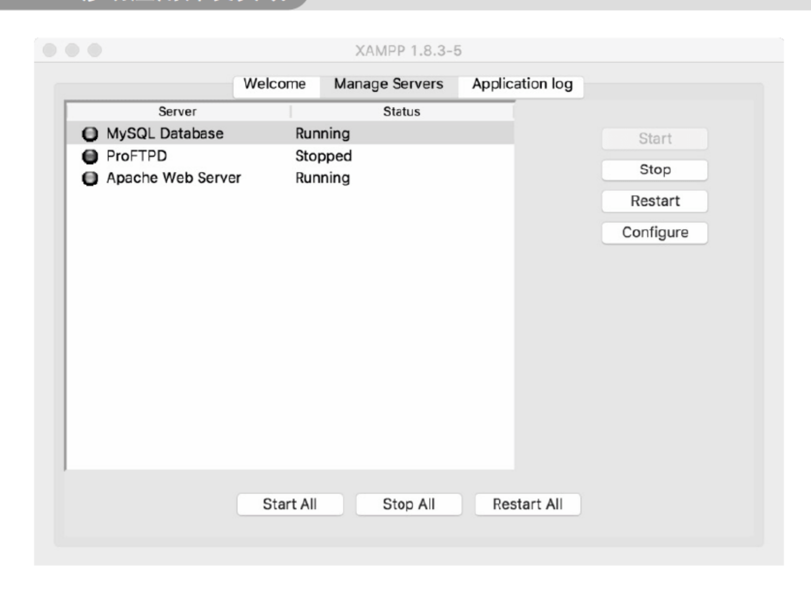


图 15-7　Xampp 的控制面板

15.4.2　JavaScript 跨域解决方法

以上我们所使用的获取 JSON 数据的方法由于跨域的原因失败了，但是实际上通过以上例子已经非常接近最终的真相了。在揭示解决方法之前笔者决定先来解释一下什么是跨域。

跨域问题是 JavaScript 安全体系中为了防止一些不安全的调用而设计的一种同源策略，具体如表 15-1 所示。

表 15-1　JavaScript 中的同源策略

URL	说明	是否允许通信
http://www.a.com/a.js　http://www.a.com/b.js	同一域名下同一目录	允许
http://www.a.com/lab/a.js　http://www.a.com/script/b.js	同一域名下不同目录	允许
http://www.a.com:8000/a.js　http://www.a.com/b.js	同一域名下不同端口	不允许
http://www.a.com/a.js　https://www.a.com/b.js	同一域名下同一目录不同协议	不允许
http://www.a.com/a.js　http://70.32.92.74/b.js	域名和域名相应的 IP	不允许
http://www.a.com/a.js　http://script.a.com/b.js	主域名和子域名	不允许
http://www.a.a.com/a.js　http://www.b.a.com/b.js	同一域名下的不同子域名	不允许
http://www.cnblogs.com/a.js　http://www.a.com/b.js	不同域名	不允许

在实际运行 Cordova 时，显然是处于不同域名下的情况（甚至就是两个没有关系的 IP），在同源策略中是不允许它们进行通信的。但是这难不倒伟大的程序员们，他们发现了许多可以突破这种限制的方法，比如范例 15-5。

【范例 15-5】使用引入的方式获取 JSON 数据。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>使用引入的方式获取 JSON 数据</title>
06  <script src="http://libs.baidu.com/jquery/2.0.0/jquery.min.js"></script>
07  </script>
08  <script type="text/javascript" src="http://127.0.0.1/test.js"></script>
09  <script>
10  // JSON 数据已经被获取，只需要将它们显示出来
11  function myAjax() {
12      $("p").append(json.firstName);
13      $("p").append("</br>");
14      $("p").append(json.lastName);
15      $("p").append("</br>");
16      $("p").append(json.age);
17      $("p").append("</br>");
18  }
19  </script>
20  </head>
21  <body>
22      <p></p>
23      <h1 onclick="myAjax();">点击这里获取数据</h1>
24  </body>
25  </html>
```

此外还需要对上传到服务器上的文件 test.js 做一些修改，修改后的内容为：

```
var json = { "firstName": "Bill","lastName": "Gates","age": 60 }
```

然后再运行本范例就可以发现已经能够实现跨域获取 JSON 数据了，如图 15-8 所示。

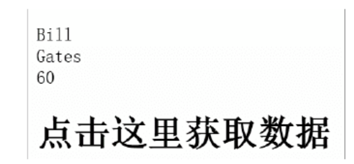


图 15-8　从服务器上获取的 JSON 数据

这样一来读者可能就明白了，只要能够在服务器端生成相应的数据，就能够很容易地利用这种方法来实现信息获取。



在实际开发中，常常使用类似的原理将一些内容进行封装，然后利用 jQuery 中的 Ajax 来提供对刷新等功能的支持。本项目主要是为了展示原理，因此不考虑这些功能。

15.4.3 服务端的实现

现在要做的就是利用 PHP 来实现服务端 JSON 数据的生成了，实际上不过是从数据库读取信息，然后对字符串做一些简单地拼接，因此很容易实现。在最开始还是要设计这次所使用的数据库。由于数据量不大，可以仅设计两个表来实现，这两个表分别记录了新闻列表、作者等内容的 news_List 表和记录了每条新闻具体内容的表 news_Neirong。它们的具体结构如表 15-2 和表 15-3 所示。

表 15-2 表 news_List 的数据结构

字段名	类型	说明
id	int	用来表示每一条新闻的 id，它的值是唯一的
author	text	用来记录新闻的作者
date	date	用来记录新闻发生的日期
title	text	新闻的标题

表 15-3 表 news_Neirong 的数据结构

字段名	类型	说明
id	int	用来表示每一条新闻的 id，它的值是唯一的
content	text	新闻的具体内容

接下来在服务器根目录下新建一个文件，命名为 test.php，按照范例 15-6 中的内容进行编辑。

【范例 15-6】利用 PHP 生成 JSON 数据。

```
01  <?php
02      // 设置页面显示的文字编码
03      header("Content-Type: text/html;charset=utf-8");
04      // 设置默认显示新闻的条数
05      $number = 20;
06      // 从 GET 参数判断是否需要对显示新闻条数进行修改
07      if(count($_GET)>0)
08      {
09          $number = $_GET['number'];
10      }
11      // 连接数据库
```

```
12      $con = mysql_connect("localhost", "root", "");
13      // 设置数据库的编码方式，一定要与数据库的编码方式相同
14      mysql_query("set names utf8");
15      // 下面一大段代码均是为了拼接出 JSON 格式的字符串并显示
16      echo "var json = [";
17      if($con)
18      {
19          // 选择要使用的数据库
20          mysql_select_db("news", $con);
21          // 数据库查询语句
22          $query = "SELECT * FROM news_List,news_Neirong WHERE
23                    news_List.id = news_Neirong.id ORDER BY news_List.id
DESC ";
24          // 通过参数设置查询数据的条目
25          $query = $query."LIMIT ".(string)$number;
26          $result = mysql_query($query);              // 执行查询操作
27          $i = 0;                                     // 用来判断是否为第一条数据
28          while($row = mysql_fetch_array($result))
29          {
30              if($i != 0)             // 如果是第一条数据，则在数据前不实现逗号分隔符
31              {
32                  echo ",";
33              }else
34              {
35                  $i = 1;
36              }
37              echo '{ "';
38              echo 'title":';
39              echo '"';
40              echo $row['title'];
41              echo '",';
42              echo '"';
43              echo 'author":';
44              echo '"';
45              echo $row['author'];
46              echo '",';
47              echo '"';
48              echo 'date":';
49              echo '"';
```

```
50                echo $row['date'];
51                echo '",';
52                echo '"';
53                echo 'content":';
54                echo '"';
55                echo $row['content'];
56                echo '"}';
57            }
58        }else
59        {
60            // 如果连接数据库失败，仍然可以返回一条 JSON 数据
61    echo '{ "title":"服务器出错了","author":"猫爷","date":"2014-04-07",
"content":"重启吧，亲！"}';
62        }
63        echo "]";
64        mysql_close($con);
65    ?>
```

保存范例文件，并在浏览器中输入地址 http://127.0.0.1/test.php，可以看到如图 15-9 所示的数据。



在测试时一定不要忘记向数据库中输入测试数据，否则得到的只能是一个空白的页面。

var json = [{ "title":"第二条新闻","author":"猫爷","date":"2014-04-07","content":"我刚刚下楼吃了碗拉面，结果发现好像泡面会更好吃一点"},{ "title":"第一条新闻","author":"猫爷","date":"2014-04-07","content":"这是第一条新闻，用于对第15章中的项目进行测试。"}]

图 15-9　利用 PHP 生成的 JSON 数据

但是现在有一个问题，在范例 15-5 中引入的是一个后缀为 js 的文件，虽然可以认为这是 JSON 的缩写，但是这并不影响浏览器将它作为标准的 JavaScript 脚本文件来解析。而在本范例中返回的地址是一个 PHP 文件，这会不会影响读取的效果呢？因此还需要使用范例 15-5 中的方法再做一次测试。

【范例 15-7】测试从 PHP 页面中获取的 JSON 数据。

```
01    <!DOCTYPE html>
02    <html>
03    <head>
04    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05    <title>使用引入的方式获取 JSON 数据</title>
06    <!-- 引入 jQuery 脚本文件 -->
07    <script src="http://libs.baidu.com/jquery/2.0.0/jquery.min.js"></script>
```

```
08  </script>
09  <!-- 引入 JSON 源数据 -->
10  <script type="text/javascript" src="http://127.0.0.1/test.php"></script>
11  <script>
12  // JSON 数据已经被获取，只需要将它们显示出来
13  function myAjax() {
14      // 遍历从 JSON 中获取的数据并将它们显示出来
15      for(i=0; i<json.length; i++) {
16          $("p").append(json[i].title);           // 显示新闻的标题
17          $("p").append(" ");
18          $("p").append(json[i].author);          // 显示新闻作者
19          $("p").append(" ");
20          $("p").append(json[i].date);            // 显示新闻日期
21          $("p").append(" ");
22          $("p").append(json[i].content);         // 显示新闻内容
23          $("p").append("</br>");                 // 最后换行结束一条新闻的显示
24      }
25  }
26  </script>
27  </head>
28  <body>
29      <p></p>
30      <h1 onclick="myAjax();">点击这里获取数据</h1>
31  </body>
32  </html>
```

运行后点击屏幕上的“点击这里获取数据”字样，可以看到确实能够从 PHP 页面上获取到数据，如图 15-10 所示。

第二条新闻 猫爷 2014-04-07 我刚刚下楼吃了碗拉面，结果发现好像泡面会更好吃一点
第一条新闻 猫爷 2014-04-07 这是第一条新闻，用于对第15章中的项目进行测试。

点击这里获取数据

图 15-10　从 PHP 页面中获取的 JSON 数据

这是为什么呢？简单地说，就是由于在范例的第 10 行中，对 script 标签进行设置时为它设置了一个属性 type="text/javascript"，这样一来浏览器进行解析的时候不管这是一个什么类型的文件都会以 JavaScript 的形式去执行其中的内容。也就相当于是在一个外部的 js（JavaScript）文件中定义了一组 JSON 数据是一样的，只不过这里的 JSON 数据是由 PHP 动态生成的。

15.5 让数据显示出来

在实现了从服务端读取数据的功能之后接下来的工作就比较轻松了，只需要将这些数据在之前写好的 HTML 页面中显示出来，然后利用 Cordova 打包就可以了。为了方便读者学习，笔者将先在范例 15-1 和范例 15-2 的基础上进行展示，然后再将完整的代码整合到范例 15-3 中实现的界面上去。首先来实现新闻列表的显示。

15.5.1 新闻列表的显示

找到范例 15-1 的文件，按照范例 15-8 对它进行修改。

【范例 15-8】新闻列表的显示。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>新闻列表的显示</title>
06  <!-- 引入 JSON 源数据 -->
07  <script type="text/javascript" src="http://127.0.0.1/test.php"></script>
08  <script>
09  function show_new(i) {
10      i++;
11      alert("这是第" + i + "条新闻");
12  }
13  </script>
14  </head>
15  <style type="text/css">
16  /* 此处省略了 CSS 样式文件，读者可参考范例15-1自行补全 */
17  </style>
18  <body>
19      <div class="main_rec">
20          <header>
21              <h1>Cordova 新闻</h1>
22          </header>
23        <div class="content" id="content_list">
24           <script>
25               for(i=0; i<json.length; i++) {
26               str = "";
27               // 生成<div class="list_rec" onClick="show_new();">
28              str = str + '<div class="';
```

```
29                  str = str + 'list_rec';
30                  str = str + '" onClick=';
31                  str = str + '"show_new(';
32                  str = str + i;
33                  str = str + ');">';
34                  // 生成<div class="list_title">
35                  str = str + '<div class="';
36                  str = str + 'list_title">';
37                  // 加入新闻标题
38                  str = str + json[i].title;
39                  // 生成</div></div>
40                  str = str + '</div></div>';
41                  document.write(str);
42              }
43          </script>
44      </div>
45    </div>
46  </body>
47  </html>
```

运行后的结果如图 15-11 所示。



图 15-11　将服务端的 JSON 数据显示在新闻列表中

可以看到，相对于范例 15-1，变化其实并不大，仅有两个主要的变化。第一是在代码的开头引入了 JSON 源，即第 7 行。其次就是在第 24~43 行使用了字符拼接得到了一段 HTML 脚本，然后利用 document.write 方法将这段脚本显示出来。

除此之外，在这段代码中还加入了一个新的自定义函数 show_new(i)。这是用来作为今后显示新闻内容时对用户点击操作预留的接口，在此可不用管它。目前，当用户点击新闻列表时，该函数会弹出一个对话框告诉用户这是第几条新闻。

此处笔者需要说明一下，为什么笔者要选用 document.write 这种方式来在页面上显示数据，而不采用 JavaScript 中的一些 DOM 方法动态地对内容进行布置？笔者这样做主要有两个原因：第一是因为在本范例中要在页面一打开时就将获取的数据显示出来，虽然说 JSON 数据量比较小，能够快速地获取到来自网络的 JSON 数据，但是毕竟是需要时间的。如果在页面一打开时就自动对数据进行动态地显示，也会导致实际上什么数据也没有获取到就显示了，这显然是不行的。另一个原因就是这样做很简单，好理解。

15.5.2 新闻内容的显示

与新闻列表相比，要在页面上显示出新闻的内容实在是太简单了，因为在这里不需要考虑一共有多少条数据。这次将在范例 15-2 的基础上进行修改，具体方法如范例 15-9 所示。

【范例 15-9】将新闻内容显示出来。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>将新闻内容显示出来</title>
06  <!-- 引入 JSON 源数据 -->
07  <script type="text/javascript" src="http://127.0.0.1/test.php"></script>
08  <script>
09      // 此标记用于记录当前显示的新闻编号
10      var news_Id = 0;
11      function show() {
12          // 显示新闻标题
13         document.getElementById("news_Title").innerHTML=json[news_Id].title;
14          // 显示新闻内容
15          document.getElementById("neirong").innerHTML=json[news_Id].content;
16          // 获取新闻作者以及发布时间信息
17          var info = "作者："+json[news_Id].author+"  发表日期：
"+json[news_Id].date;
18          // 将新闻作者以及发布时间等信息显示出来
19          document.getElementById("news_Info").innerHTML=info;
20      }
21  </script>
22  </head>
23  <style type="text/css">
24  /* 此处省略了 CSS 样式文件，读者可参考范例15-1自行补全 */
25  </style>
```

```
26 <body>
27     <div class="main_rec" onClick="show();">
28         <header>
29             <div id="back">返回</div>
30             <h1>Cordova 新闻</h1>
31         </header>
32         <div class="content">
33             <h1 id="news_Title"></h1>
34             <h4 id="news_Info">作者：  发表日期：</h4>
35             <div id="line"></div>
36             <div id="neirong">
37             </div>
38         </div>
39     </div>
40 </body>
41 </html>
```

需要注意的是，在运行本范例时需要先点击一下屏幕才能获取到，如图 15-12 所示，这也是由于如果直接获取显示数据的操作可能在获取数据之前实现而产生错误所采取的变通之法。乍一看这可能会让某些用户觉得很不爽，但是再一想这个页面本来就是需要通过点击来实现的，因此倒是没有什么需要担忧的了。

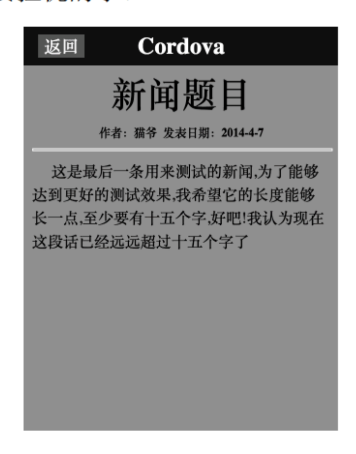


图 15-12　单击进入新闻详情页面

15.5.3　最终的整合

至此，该项目的全部功能就已经实现了，现在要做的就是将这两个页面结合在一起，请各位读者找到范例 15-3 的代码，参照本节介绍的范例 15-8 和范例 15-9 进行修改。修改后的内容如范例 15-10 所示。

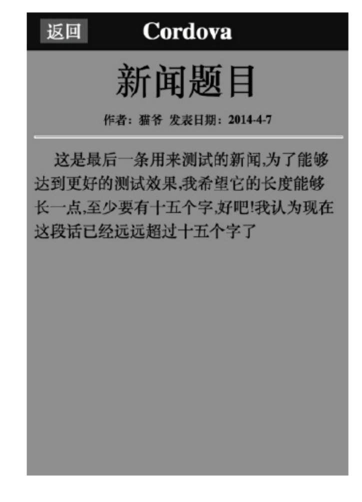
【范例 15-10】最终实现的新闻客户端页面

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>最终实现的新闻客户端页面</title>
06  <!-- 引入 JSON 源数据 -->
07  <script type="text/javascript" src="http://127.0.0.1/test.php">
08  </script>
09  <script>
10  // 用来从新闻内容界面返回到新闻列表
11  function show_list() {
12      var list = document.getElementById("list");
13      var news = document.getElementById("news");
14      news.style.display = "none";
15      list.style.display = "block";
16  }
17  // 从新闻列表切换到新闻内容页
18  function show_news() {
19      var list = document.getElementById("list");
20      var news = document.getElementById("news");
21      list.style.display = "none";
22      news.style.display = "block";
23  }
24  //  用户点击新闻列表项后触发此操作，实现的功能是显示相应的新闻内容
25  function show_new(i) {
26      // 显示新闻内容页
27      show_news();
28      // 将新闻内容加载到页面上
29      show(i);
30  }
31  // 将相应的新闻内容加载到屏幕上，其中 id 的值是新闻编号
32  function show(id) {
33      // 显示新闻标题
34      document.getElementById("news_Title").innerHTML=json[id].title;
35      // 显示新闻内容
36      document.getElementById("neirong").innerHTML=json[id].content;
37      // 获取新闻作者以及发布时间信息
```

```
38      var info = "作者："+json[id].author+"  发表日期：
"+json[id].date;
39      // 将新闻作者以及发布时间等信息显示出来
40      document.getElementById("news_Info").innerHTML=info;
41      }
42  </script>
43  </head>
44  <style type="text/css">
45  /* 此处省略了 CSS 样式文件，读者可参考范例15-1自行补全 */
46  </style>
47  <body>
48      <div class="main_rec" id="news" style="display:none;">
49          <header>
50              <div id="back" onClick="show_list();">返回</div>
51              <h1>Cordova 新闻</h1>
52          </header>
53          <div class="content">
54              <h1 id="news_Title">新闻题目</h1>
55              <h4 id="news_Info">作者：猫爷  发表日期：2014-4-7</h4>
56                <div id="line"></div>
57             <div id="neirong">正文内容</div>
58          </div>
59      </div>
60      <!--上面部分是新闻内容，下面部分是新闻列表-->
61      <div class="main_rec" id="list">
62          <header>
63              <h1 onClick="show_news();">Cordova 新闻</h1>
64          </header>
65          <div class="content">
66          <script>
67             for(i=0; i<json.length; i++) {
68                str = "";
69                // 生成<div class="list_rec" onClick="show_new();">
70                str = str + '<div class="';
71                str = str + 'list_rec';
72                str = str + '" onClick=';
73                str = str + '"show_new(';
74                str = str + i;
75                str = str + ');">';
```

```
76                // 生成<div class="list_title">
77                str = str + '<div class="';
78                str = str + 'list_title">';
79                // 加入新闻标题
80                str = str + json[i].title;
81                // 生成</div></div>
82                str = str + '</div></div>';
83                document.write(str);
84            }
85        </script>
86      </div>
87    </div>
88  </body>
90  </html>
```

这样一来就可以一个独立应用的形式运行了。双击之后显示的是新闻列表，如图 15-13 所示，点击任意一个列表项则会切换到相应的新闻内容界面，如图 15-14 所示，再点击“返回”按钮又将回到新闻列表界面。



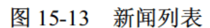


图 15-13　新闻列表　　　　图 15-14　新闻内容界面

下面笔者再来讲解一下实现的思路（不是代码的原理，而是怎样由范例 15-8 和范例 15-9 得到该范例）。首先，在范例 15-3 中已经准备好了两个界面之间切换的功能，但是它在内容的显示方面仍然是“静态的”，即无法从服务端获取数据。因此第一部就是引入外部的 JSON 数据，如范例第 7 行所示。

接下来就是将这些数据在列表中显示，如范例第 66~85 行所示，与范例 15-8 中完全一样，直接复制就可以了。由于此处引用了一个自定义函数 show_new()，因此需要把该函数一起复制过来。但是此时该函数就应当能够发挥一些作用了，比如页面间的跳转。

除此之外，由于新闻编号在该自定义函数中被当作参数传递，因此可以通过该函数来实现新闻内容的加载（范例第 32~41 行）。



由于在此处使用 innerHTML 方法会直接替换掉之前写在元素中的内容，因此不需要考虑当第二次打开新闻内容时内容会不会重叠的问题。

最后，就可以将整个 HTML 文件复制到 Cordova 中进行测试了，不过还是要提醒读者，如果是在真机上进行测试的话，不要忘记连接网络。还有，如果是使用一键安装的 Apache，可能默认是不能被外网访问的，需要在 httpd-xampp.conf 文件中找到一行内容为 Deny from all 的语句用“#”注释掉。

15.6 小结

到此为止，一款具有一定实用价值的新闻客户端就算是做好了，而且笔者甚至可以毫不客气地说，本章所使用的这个例子要比许多真正上线的原生安卓新闻应用要“好用”得多，而且它实现起来非常简单，最重要的是除了最终的测试，其他所有步骤笔者都是通过 PC 端的浏览器来进行的，这使得笔者在开发时不必对着那个令人懊恼的模拟器较劲了。如果读者对它有更高的期望就可以尝试为它加入一些更加人性化的功能，比如可以在新闻中插入图片使内容更加丰富，也可以从网上寻找一些具有下拉刷新功能的插件加入到列表中，使用户能够以更加舒适的方式获取到信息。随着新加入的功能越来越多，这款简单的“客户端”最终很可能会成为一款具有强大功能与交互性的真正的大型新闻客户端。

第 16 章 实战Cordova制作号码本

前面两章已经介绍过两个利用 Cordova 实际开发应用的例子，但是这两个例子作为 Cordova 教程中的实例都有一个致命的缺点，就是没有真正用到 Cordova 中的 API。本章所介绍的实例将弥补这个不足，利用 Cordova 中的联系人对象制作一款电话号码本。此外本章还要补充一个在之前的内容中遗漏的重要知识点，那就是在 Cordova 中自定义插件的方法，本章将利用此方法来实现 Cordova 的电话拨号功能。

另外，本章将使用 jQuery Mobile 来生成应用的界面，读者不用再忍受笔者制作的那种简陋而且难看的界面了。

本章的主要知识点有：

- 使用 jQuery Mobile 来实现界面，以及利用 JavaScript 动态加载页面中的内容。
- 如何将 Cordova 中读取的联系人信息加载到应用中。
- Cordova 插件的制作方法。

16.1 项目介绍

本章将要模仿实现一款简单的电话号码本的功能，包括了对联系人列表的查看、新建联系人、删除联系人以及拨打电话、发送短信等功能。在设计这款应用之前可以参考安卓系统自带的联系人界面，如图 16-1 所示。

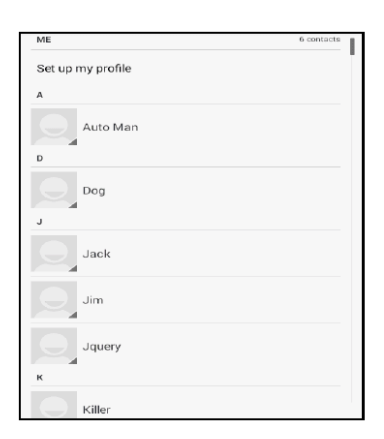


图 16-1　安卓自带联系人界面

可以通过“新建联系人”进入如图 16-2 所示的界面，用户可以在其中创建新的联系人。当然也可以点开某个具体的联系人来对联系人的资料进行编辑或查看，如图 16-3 所示。



图 16-2　新建联系人的界面

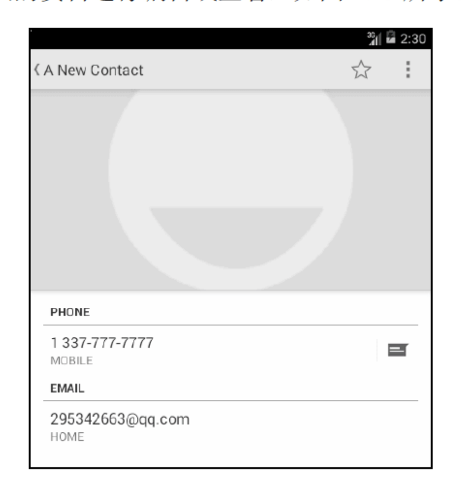


图 16-3　可在该界面查看某联系人的具体信息

此外，还可以直接通过联系人的界面对联系人进行拨号；而在本章中就将要实现一款有类似功能的应用，但是要根据实际需要简单地调整。

16.2 为 Cordova 编写插件

通过之前的分析，相信读者已经对本章的项目有了大概了解，然而本项目有一个重要的功能是利用本章实现的 APP 完成发送短信和拨打电话的功能，可是通过本书的学习，读者应该已经了解到在 Cordova 中并没有提供实现这两个功能的 API。当然可以用一些办法来糊弄用户，比如用户点击代表“发送”功能的按钮后弹出一个对话框“对不起，该功能正在开发中，敬请期待”。但是历史告诉笔者这绝对会在招来用户的咒骂之前，先招来读者的咒骂，因此只能寻找其他办法。

笔者一直坚信拿来主义，有现成的为什么不用？但是当没有现成的工具可以使用时，笔者也不会随意向需求认输。本节将亲自来实现利用 Cordova 发送短信和打电话的功能插件，来弥补 Cordova 的不足。

16.2.1　实现发短信的插件

本小节首先详细地介绍了 Cordova 开发短信插件的方法，也顺便让读者了解是怎样为 Cordova 加入自己需要的 API 的。

一般标准的 Cordova 的插件的机构如图 16-4 和图 16-5 所示。



图 16-4　插件目录 1

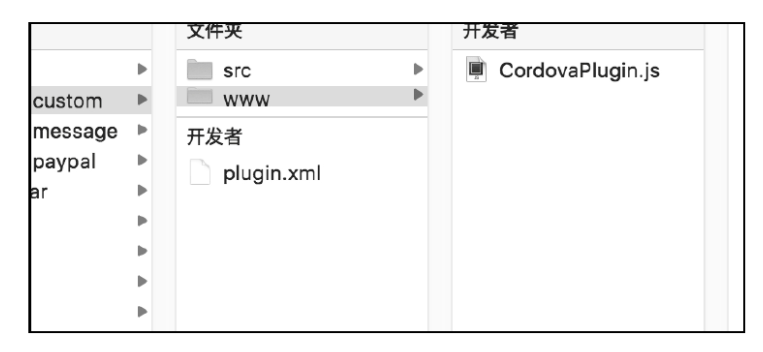


图 16-5　插件目录 2

目录中的 CordovaPlugin.js 是用来连接 Cordova 和 iOS 原生代码的文件，代码如下：

```
1 var exec = require('cordova/exec');

2  var CordovaPlugin=function(){
3     }
4  CordovaPlugin.prototype.theplatform="android | ios";
5 CordovaPlugin.prototype.send =
function(successCallback,errorCallBack,target,content){
6      exec(successCallback,
7              errorCallBack,
8           "CordovaPlugin",
9           "send",
10           [target,content]
11        );
12       }
13 CordovaPlugin.install = function () {
14  if (!window.plugins) {
15   window.plugins = {};
16  }
```

```
17  window.plugins.CordovaPlugin = new CordovaPlugin();
18  return window.plugins.Message;
19 };
20 cordova.addConstructor(CordovaPlugin.install);
```

有了这一段代码，就可以理解整个插件的原理了。

在 Cordova 提供的大多数 API 中都会用到 success 和 error 作为参数分别用来调用 API 使用成功和失败。除此之外，该 API 应该还有两个参数，分别保存短信发送的目标号码和短信的内容。

exec()方法的参数是不是和定义的 API 参数非常类似，只是多了一个值为 send 的参数。再回头看范例 16-1 的第 18 行，就是将接收到的来自 HTML 页面的参数与常量 SEND 做比较。而常量 SEND 的值又恰恰是 send。

这下应该明白了吧，exec 是 Cordova 中定义的一个方法，它能够将一些参数发送给系统，在本例中被类 Message 接收并处理。有读者可能要问为什么系统会知道这个方法要由 Message 类来处理呢？因为在 exec()方法的第三个参数中已经说明了处理这个请求的类名为 Message。

plugin.xml 包含了插件的配置信息，里面的代码如下：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- Copyright (c) 2014 PayPal. All rights reserved. -->
3 <plugin xmlns="http://apache.org/cordova/ns/plugins/1.0"
4   xmlns:rim="http://www.blackberry.com/ns/widgets"
5   xmlns:android="http://schemas.android.com/apk/res/android"
6   id="cordova-plugin-custom"
7   version="0.0.1">
8   <name>message</name>
9   <description>send message</description>
10   <keywords>CordovaPlugin, sdk</keywords>
11   <license>BSD License, see LICENSE.md for details</license>
12   <engines>
13         <engine name="cordova" version="&gt;=3.0.0" />
14   </engines>

15   <js-module src="www/CordovaPlugin.js" name="CordovaPlugin">
16       <clobbers target="cordova.plugin.message" />
17   </js-module>
18  <!-- ios -->
19   <platform name="ios">
20       <config-file target="config.xml" parent="/*">
21           <feature name="CordovaPlugin">
22               <param name="ios-package" value="CordovaPlugin" onload="true" />
```

```
23          </feature>
24       </config-file>
25       <header-file src="src/ios/CordovaPlugin.h" />
26       <source-file src="src/ios/CordovaPlugin.m" />
27    </platform>
29  </plugin>
```

CordovaPlugin.h 和 CordovaPlugin.m 文件是 iOS 的原生文件，可以通过下面方法生成。

首先在项目中新建一个类，命名为 CordovaPlugin，如图 16-6 所示，并让它继承类 CDVPlugin，在其中编写代码，如下所示。

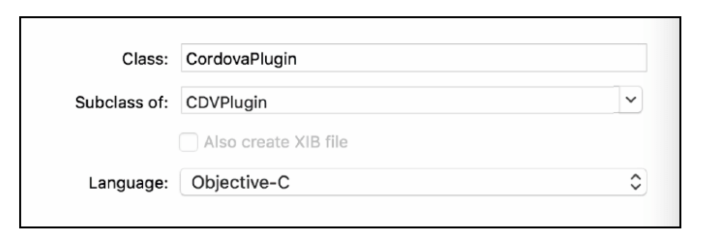


图 16-6　在 xcode 里面添加插件所需原生文件

CordovaPlugin.h 的声明：

```
1  #import <Cordova/CDVPlugin.h>
2 #import <MessageUI/MessageUI.h>
3 #import <MessageUI/MFMessageComposeViewController.h>
4 @interface CordovaPlugin : CDVPlugin
5 -(void)send:(CDVInvokedUrlCommand *)command;
6 @end
```

Message.m 的实现：

```
1 #import "CordovaPlugin.h"
2 #import "AppDelegate.h"

3 @interface CordovaPlugin()

4 @end

5 @implementation CordovaPlugin

6 -(void)send:(CDVInvokedUrlCommand *)command
7 {
8   [self.commandDelegate runInBackground:^{
```

```
9     CDVPluginResult* pluginResult = nil;
10      NSString* phoneNumber = [command.arguments objectAtIndex:0];
11      NSString* textMessage = [command.arguments objectAtIndex:1];
12     if (![MFMessageComposeViewController canSendText]) {
13                NSMutableDictionary* returnInfo = [NSMutableDictionary
dictionaryWithCapacity:2];
14      [returnInfo setObject:@"SMS_FEATURE_NOT_SUPPORTED" forKey:@"code"];
15         [returnInfo setObject:@"SMS feature is not supported on this device"
forKey:@"message"];

16   pluginResult = [CDVPluginResult resultWithStatus:CDVCommandStatus_ERROR
messageAsDictionary:returnInfo];

17         [self.commandDelegate sendPluginResult:pluginResult
callbackId:command.callbackId];
18         return;
19     }
20      MFMessageComposeViewController *composeViewController =
[[MFMessageComposeViewController alloc] init];
21      composeViewController.messageComposeDelegate = self;
22      NSMutableArray *recipients = [[NSMutableArray alloc] init];
23      [recipients addObject:phoneNumber];
24      [composeViewController setBody:textMessage];
25      [composeViewController setRecipients:recipients];
26      dispatch_async(dispatch_get_main_queue(), ^{
27         [self.viewController presentViewController:composeViewController 28
animated:YES completion:nil];
29      });
30   }];
31 }
32 -(void)messageComposeViewController:(MFMessageComposeViewController
*)controller didFinishWithResult:(MessageComposeResult)result
33 {
34   [self.viewController dismissViewControllerAnimated:YES completion:nil];
35   switch (result) {
36      case MessageComposeResultSent:
37         //信息传送成功
38         NSLog(@"消息发送成功");
39         break;
```

```
40        case MessageComposeResultFailed:
41            //信息传送失败
42            NSLog(@"消息发送失败");
43
44            break;
45        case MessageComposeResultCancelled:
46            //信息被用户取消传送
47            NSLog(@"取消发送");
48            break;
49        default:
50            break;
51    }
52 }
53@end
```

之后保存编写的代码。



在某些版本的 Cordova 中也许需要继承 Plugin 类或者是其他有类似名称的类，这是由于 Cordova 版本不同造成的。

最后在命令行中运行指令添加插件：

```
cordova plugin add 本地插件路径
```

接下来就可以在手机上测试这个插件是不是真的有效了，使用方法如范例 16-1 所示。

【范例 16-1】使用 Cordova 发送短信。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>使用 jQuery Mobile 实现短信编辑界面</title>
06  <meta name="viewport" content="width=device-width, initial-scale=1">
07  <!—-引入 Cordova 所需的脚本文件 -->
08  <script src="js/cordova.js" type="text/javascript"></script>
09  <script>
10  function send() {
11      // 使用发送短信的 API 发送短信给另一台虚拟机，其中5556是另一台虚拟机的号码
12      window.plugins.CordovaPlugin.send(success, error, "10010", "1"
13              );
14  }
15  function success() {
```

```
16      alert("短信发送成功");
17  }
18  function error() {
19      alert("短信发送失败");
20  }
21  </script>
22  </head>
23  <body>
24      <h1 onClick="send();">发送短信</h1>
25  </body>
26 </html>
```

连接手机配置好相应的证书后，编译运行，点击发送短信，就可以调整到手机短信发送页面；而且，页面已经填写了收件人，只需要添加内容就可以了，如图 16-7 所示。



图 16-7　跳转到短信发送页面

细心的话会发现点击发送后，会直接返回到我们的 demo 页面。

16.2.2　为 Cordova 编写电话拨号插件

以上实现了一个在 Cordova 中不具备的功能——短信发送，本小节来实现利用 Cordova 打电话的插件。

【范例 16-2】在 Message.m 中添加实现 call 方法的代码。

```
-(void)call:(CDVInvokedUrlCommand *)command
{
   NSString *phoneNumber = [command.arguments objectAtIndex:0];
```

```
    NSMutableString * str=[[NSMutableString alloc]
initWithFormat:@"telprompt://%@",phoneNumber];

    [[UIApplication sharedApplication]openURL:[NSURL URLWithString:str] options:@{}
completionHandler:^(BOOL success) {

    }];

}
```

接下来再编写对应的 JavaScript 文件，直接在 CordovaPlugin.js 中加入以下内容：

```
1 CordovaPlugin.prototype.call = function(successCallback,errorCallBack,target){
2       exec(successCallback,
3               errorCallBack,
4           "CordovaPlugin",
5           "call",
6           [target]
7       );
8     }
```

由于代码与前面所讲类似，这里就不再做讲解了，请读者自行对比代码进行理解。

16.3 界面设计

以上通过编写插件的方式实现了利用 Cordova 拨打电话和发送短信的功能，而 Cordova 本身又提供了对联系人信息的获取功能，也就是说目前想要实现这款号码本应用的全部功能都是可以的了。此时就可以放心大胆地进行界面的设计了。

我们要实现的主要功能就是对联系人的创建、删除以及通过联系人来进行打电话或发送短信的操作。

该应用最主要的界面应当是一个列表，观察图 16-1 中的界面会发现安卓自带的联系人列表中并没有列出联系人的号码，笔者刚发现这一点的时候觉得有些不可思议，但当笔者经过了仔细思考后终于发现这种做法在某种程度上是非常合情合理的。

因为对用户来说只要知道电话将拨给哪个联系人即可，而不需要知道这个人的号码具体是多少，这对于手机屏幕上有限的空间来说是非常有必要的。因此在本项目中也不会将联系人的号码显示在联系人列表中。

此外，在本项目中将取消列表中的联系人头像，毕竟现在越来越多的人开始愿意购买一部廉价的安卓手机作为备用机使用。他们只需要这部手机能够打电话就可以了，而且许多人也懒得为每一个联系人加入照片头像，但是每次打电话时看着空白的头像又会感到别扭，索性笔者就取消这一功能好了。

除此之外还要在界面的底部保留让用户添加联系人的按钮，最终设计出的联系人列表如图 16-8 所示。其中，顶部栏仅作装饰使用，也可以根据需要加入“退出”按钮，底部栏是一个按钮，用于新建联系人使用。顶部栏偏下的地方是一个搜索栏，用于查找联系人使用，而页面的主要空间则被联系人列表占据。列表中的每一项分为两部分，其中左侧显示联系人的姓名或是昵称，点击之后则会自动向该联系人拨打电话；右侧是一个图标，点击之后则会跳转到短信发送页面。

由此可知，该项目中还需要短信发送和新建联系人两个界面，分别如图 16-9 和图 16-10 所示。

其中，图 16-9 展示的是短信编辑和发送的界面，可以在其中编辑短信内容并通过底部的“发送”按钮发送短信，也可以点击左上角的“返回”按钮取消短信编辑并返回到联系人列表中去。除此之外，还应当能将短信发送的目标在顶部栏中显示出来，比如当笔者给 10086 发送短信时就应当显示出“发送给：10086”。



图 16-8 设计好的联系人列表

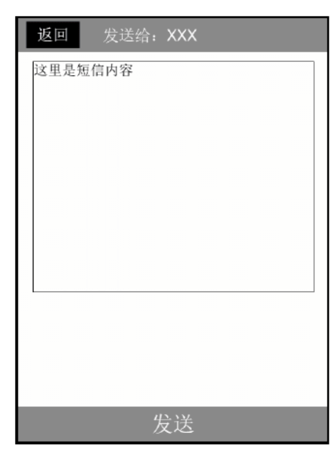


图 16-9 短信编辑和发送界面设计

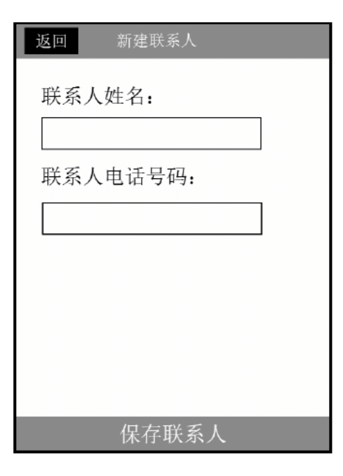


图 16-10 新建联系人界面设计

图 16-10 展示出的是该项目新建联系人的界面，对比图 16-2 中的安卓原生新建联系人界面会发现，笔者设计的界面在功能上要简单了许多，许多功能（比如邮箱和地址）都被笔者有意识地省略了。读者可以思考一下，日常生活中有多少人能够使用到这些选项呢？最简单的往往也是最方便的。

与笔者在图 16-9 中所展示的短信编辑界面类似，用户也可以通过左上角的“返回”按钮取消当前的操作，而短信编辑界面底部的“发送”按钮在“新建联系人”界面中则被替换成了“保存联系人”，用户可以通过它来保存输入的联系人以及号码。



在完成这三个界面的设计之后笔者突然有一个发现，那就是笔者这次所设计出的界面远远超出了笔者的美工水平，不但配色精美而且线条简洁。但是由于本章要介绍在项目中使用 jQuery Mobile 的方法而得不到使用，这实在是有一些遗憾。

读者在实际开发中也常常会遇到这样的问题，原本已经设计好了一套方案（并不局限于界面设计），但是当进度已经进行了一半时突然发现了一个更好的想法，这时可能就会纠结要不要更改方案。

本书由于有要求必须使用 jQuery Mobile，并且笔者也没有考虑过将这款应用推向市场，因此不需要为此而纠结；而对于读者来说，遇到类似的情况首先要考虑时间上是否允许，而在时间允许的情况下则要尽可能地将两种方案不同的地方分别实现再进行对比。如果时间上仅够实现某一种方案，那么笔者建议还是坚持原有方案，在有余力的情况下再去尝试新的方案。

16.4 界面的实现

以上已经设计好了本章所要实现应用的界面布局，本节要做的就是利用 jQuery Mobile 强大的 UI 控件，让这种布局以一套完整界面的形式展示出来。

16.4.1 联系人列表的实现

首先按照图 16-8 中所设计的布局，利用 jQuery Mobile 实现联系人列表，具体的实现过程如范例 16-3 所示。

【范例 16-3】使用 jQuery Mobile 实现联系人列表。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>使用 jQuery Mobile 实现联系人列表</title>
06  <meta name="viewport" content="width=device-width, initial-scale=0.5">
07  <!-- 引入 jQuery Mobile 所需的 CSS 样式文件 -->
08  <link rel="stylesheet"
href="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css" />
09  <!-- 引入 jQuery 脚本文件用来提高对 jQuery Mobile 的支持 -->
10  <script src="http://code.jquery.com/jquery-1.11.1.min.js"></script>
11  <!-- 引入 jQuery Mobile 所需的脚本文件 -->
12  <script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-
1.4.5.min.js"></script>
13  </head>
14  <body>
15      <div data-role="page" data-theme="a">
16          <div data-role="header" data-position="fixed">
17            <h1>联系人列表</h1>
18          </div>
19         <!--此处为列表控件 -->
20          <ul id="mylist" data-role="listview" data-split-icon="gear" data-
filter="true">
21              <li>
22                  <a>联系人</a>
```

```
23                    <a></a>          <!—-空白的 a 标签用于显示列表项右侧的按钮 -->
24                </li>
25                <!--此处省略多行，内容与本范例第21~24行完全一致，读者可自行补齐-->
26            </ul>
27            <div data-role="footer" data-position="fixed">
28                <div  data-role="navbar" data-position="fixed">
29                    <ul>
30                        <li><a><h2>新建联系人</h2></a></li>
31                    </ul>
32                </div>
33            </div>
34        </div>
35    </body>
36    </html>
```

运行后的界面如图 16-11 所示。下面对本次使用的范例代码进行讲解。

图 16-11　实现后的联系人列表界面

首先要明确的一点是，由于不需要开发者自己编写 CSS，并且各个元素之间的显示关系都被 jQuery Mobile 做出了完整的定义，使得开发者不需要再编写大量的代码了。比如当页面中使用顶部栏时只要为 div 元素加入属性 data-role="header"，那么该元素就会显示在屏幕的最顶端，而不需要额外对它的位置进行定义。

在代码的开始部分，首先要将需要的 jQuery Mobile 的脚本和 CSS 样式文件引入到代码中，如范例第 8、10、12 行所示，然后就可以在 body 标签中开始加入 page 元素，如范例第 15 行所示。

本范例使用了顶部栏和尾部栏，分别为 div 标签加入属性 data-role="header" 和 data-role="footer"来声明。为了保证它们始终位于屏幕的顶部和底部，并且不会对列表中的内容形成遮挡，需要对它们加入属性 data-position="fixed"，如范例第 16 行和第 27 行所示。

然后就可以向页面中加入列表了，在 jQuery Mobile 中，如果希望向页面中加入某个控件只需要为相应的 div 标签指定它的 data-role 属性即可，如范例第 28 行所示。考虑到用户如果保存

了许多联系人，在查找时可能会不方便，有时需要通过一个搜索栏来帮助用户查找相应的联系人，那么可以通过为列表控件加入属性 data-filter="true"来为该列表加入一个具有动态查找功能的搜索框。

之后就可以按照范例第 21~24 行的样式为列表中加入内容了，此处为了节约篇幅省略了一些重复的内容，读者可以自行加入进去。不过如果懒得加入内容而完全照抄本范例的代码也并不影响使用。

16.4.2 新建联系人界面的实现

实现了联系人列表之后，再来实现一个新建联系人的界面，新建一个 HTML 文件并命名为 create.html，按照范例 16-4 所示输入代码。

【范例 16-4】使用 jQuery Mobile 实现新建联系人界面。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>使用 jQuery Mobile 实现新建联系人界面</title>
06  <meta name="viewport" content="width=device-width, initial-scale=1">
07  <!-- 引入 jQuery Mobile 所需的 CSS 样式文件 -->
08  <link rel="stylesheet"
href="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css" />
09  <!-- 引入 jQuery 脚本文件用来提高对 jQuery Mobile 的支持 -->
10  <script src="http://code.jquery.com/jquery-1.11.1.min.js"></script>
11  <!-- 引入 jQuery Mobile 所需的脚本文件 -->
12  <script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-
1.4.5.min.js"></script>
13  </head>
14  <body>
15      <div data-role="page" data-theme="a">
16          <div data-role="header" data-position="fixed">
17              <a href="index.html" data-role="button">返回</a>
18                <h1>新建联系人</h1>
19         </div>
20          <div data-role="content">
21              <label for="name">联系人姓名:</label>
22              <input type="text" id="name" placeholder="请输入联系人姓名" />
23             <label for="number">联系人号码:</label>
24              <input type="tel" id="number" placeholder="请输入联系人号码"/>
```

```
25          </div>
26            <div data-role="footer" data-position="fixed">
27                <div  data-role="navbar" data-position="fixed">
28                    <ul>
29                        <li><a href="#"><h2>创建联系人</h2></a></li>
30                    </ul>
31                </div>
32          </div>
33      </div>
34  </body>
35  </html>
```

运行之后界面如图 16-12 所示。

图 16-12　新建联系人界面

可以通过界面左上角的“返回”按钮取消新建联系人的操作，也可以点击底部的“创建联系人”来保存填写好的联系人信息。

通过代码可以看出，本例的整体结构与范例 16-3 非常类似，只是将列表控件取消并加入了一个名为 content 的元素，见范例 16-4 第 20~25 行所示，其中包含了两个标签和两个文本编辑框。通过范例 16-4 中的代码可以看出，label 声明了标签控件，可以通过设置它的 for 属性使之与和它具有相同 id 的编辑框对应。另外，为文本编辑框指定属性 placeholder，可以设置该编辑框中的提示字符属性，如图 16-12 中可以看到编辑框中有“请输入联系人姓名”字样。

16.4.3　短信编辑界面的实现

本节中将继续实现短信编辑界面，实际上本小节的任务非常简单，只需要在范例 16-6 的基础上稍作修改就可以了。新建一个文件命名为 send.html，先将范例 16-4 中的内容复制过来，再

按照范例 16-5 中的内容对它进行修改。

【范例 16-5】使用 jQuery Mobile 实现短信编辑界面。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>使用 jQuery Mobile 实现短信编辑界面</title>
06  <meta name="viewport" content="width=device-width, initial-scale=1">
07  <!--引入 jQuery Mobile 的样式文件 -->
08  <link rel="stylesheet" href="jquery.mobile.min.css" />
09  <!--引入 jQuery 脚本文件 -->
10  <script src="jquery-1.7.1.min.js"></script>
11  <!--引入 jQuery Mobile 的脚本文件 -->
12  <script src="jquery.mobile.min.js"></script>
13  </head>
14  <body>
15      <div data-role="page" data-theme="a">
16          <div data-role="header" data-position="fixed">
17              <a href="index.html" data-role="button">返回</a>
18              <h1>发送给: XXX</h1>
19          </div>
20          <div data-role="content">
21              <label for="message">编辑短信内容:</label>
22              <textarea id="message"></textarea>
23         </div>
24          <div data-role="footer" data-position="fixed">
25              <div  data-role="navbar" data-position="fixed">
26                  <ul>
27                      <li><a href="#"><h2>发送</h2></a></li>
28                  </ul>
29              </div>
30         </div>
31     </div>
32  </body>
33  </html>
```

是不是与范例 16-4 的内容非常相似呢？甚至是更加简单了，运行之后的界面如图 16-13 所示。由于过于简单，笔者这里就不再对范例做太多介绍了。



图 16-13　发送短信界面

但是在 iOS 的开发中，因为相关权限问题，开发在调用系统短信发送功能的时候是必须调整到苹果提供的界面的，对安卓开发感兴趣的同学可以尝试上面的界面。

16.5 界面功能的实现

以上已经实现了在 16.3 节中设计的界面，然而这还不够，因为在以上设计的联系人列表中的联系人是固定的，而在实际使用中的联系人列表是由 Cordova 在用户的手机中获取的。这就导致了在范例 16-3 中所实现的界面是无法拿来直接使用的，而本节的任务就是对它做一些修改，使它变成一套真正有用的界面。

16.5.1　联系人数据的生成

在进行对界面的修改之前需要先实现一段简单的脚本，用于生成一些简单的联系人数据，这样一来不但测试更加方便，而且只需要在最后一步时加入使用 Cordova 获取的联系人信息，就可以直接使用了。

还记得在前面实现的新闻应用吗？可以说整个过程都是在 PC 端实现的，只有最后才需要在虚拟机或者是真机上进行测试。笔者非常享受这样的开发过程，因为这比完全在虚拟机中测试要有效率得多。本章的项目由于要使用到一些 API 而不得不在开发中使用虚拟机测试，但是如果能够将不需要 API 的部分与需要 API 的部分尽量地分离，使得大多数工作能够在 PC 端进行测试，这无疑也是很好的。

由于 JavaScript 并不是一种真正面向对象的语言，而是一种基于面向对象思想的脚本语言，因此没有办法去实现一个专门的类用于保存联系人信息，但是却可以当作真实的有这样的类存在，只要按照一定的语法使用就可以了。



事实上在本书前面的内容中也常常提到 Cordova 封装了某个类（比如 Contact 类），但是实际上这些类也是根本不存在的，但是由于 JavaScript 的特性却可以让用户像真正存在这些类一样对对象进行操作。

在本项目中需要一个类 Person，在 Person 类中封装了联系人的姓名、号码等信息，在具体使用时可以按照如下所示的方法来实现：

```
var people = new Array();                         // 新建一个数组用于存放联系人信息
for( i=0; i<10; i++) {
    // 在实际开发中，将这部分换成利用 API 读取联系人信息。
    people[i] = new Object();                     // 将数组元素类型转化为一个对象
    people[i].id = i;
    people[i].name = "XXX";
    people[i].number = "XXXXXXXXXXX"
}
for(i=0; i<people.length; i++) {
    alert("第" + people[i].id + "个联系人是：" + people[i].name
+  "号码是:" + people[i].number);
}
```

下面开始利用这段代码让联系人信息“动态地”加载到 jQuery Mobile 所实现的联系人列表上去，具体实现方法如范例 16-6 所示。

【范例 16-6】让联系人动态地显示在联系人列表。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05  <title>使用 jQuery Mobile 实现联系人列表</title>
06  <meta name="viewport" content="width=device-width, initial-scale=0.5">
07  <!-- 引入 jQuery Mobile 所需的 CSS 样式文件 -->
08  <link rel="stylesheet" href="jquery.mobile.min.css" />
09  <!-- 引入 jQuery 脚本文件用来提高对 jQuery Mobile 的支持 -->
10  <script src="jquery-1.7.1.min.js"></script>
11  <!-- 引入 jQuery Mobile 所需的脚本文件 -->
12  <script src="jquery.mobile.min.js"></script>
13  <script>
14  var people = new Array();                     // 新建一个数组用于存放联系人信息
15  function show_List() {
```

```
16       for( i=0; i<10; i++) {
17       // 在实际开发中，将这部分换成利用 API 读取联系人信息。
18           people[i] = new Object();                // 将数组元素类型转化为一个对象
19           people[i].id = i;                         // 获取每个联系人的 id
20           people[i].name = "联系人" + i;            // 获取每个联系人的姓名
21           people[i].number = "0411" + i + i;        // 获取每个联系人的号码
22       }
23       for(i=0; i<people.length; i++) {
24           // 加入内容 <li><a onclick="call(i);">联系人</a><a
onclick="message(i);">
</a></li>
25           var str = ' <li>';
26           str = str + '<a onclick=\"call(' + i + ');\">';
27           str = str + people[i].name;
28           str = str + '</a>';
29           str = str + '<a onclick=\"message(' + i + ');\">';
30           str = str + '</a></li>';
31           $("#mylist").append(str);
32           $("#mylist").listview("refresh");
33       }
34   }
35   // 当程序完成时，在此处调用打电话的 API
36   function call(i) {
37       i++;
38       var str = "你想给第" + i + "个人打电话,";
39       str = str + "他叫: " + people[i-1].name + ",";
40       str = str + "他的电话号码是:" + people[i-1].number + "。";
41       alert(str);
42   }
43   // 当程序完成时，在此处调用发短信的 API
44   function message(i) {
45       i++;
46       var str = "你想给第" + i + "个人发短信,";
47       str = str + "他叫: " + people[i-1].name + ",";
48       str = str + "他的电话号码是:" + people[i-1].number + "。";
49       alert(str);
50   }
51   </script>
52   </head>
```

```
53  <body onLoad="show_List();">
54      <div data-role="page" data-theme="a">
55          <div data-role="header" data-position="fixed">
56            <h1>联系人列表</h1>
57         </div>
58         <!-- 这里是联系人列表 -->
59          <ul id="mylist" data-role="listview" data-split-icon="gear" data-
filter
="true">
60              <!-- 新加入的内容将在这里被显示出来 -->
61          </ul>
62          <div data-role="footer" data-position="fixed">
63              <div  data-role="navbar" data-position="fixed">
64                  <ul>
65                      <li><a><h2>新建联系人</h2></a></li>
66                  </ul>
67              </div>
68         </div>
69      </div>
70  </body>
71  </html>
```

运行之后的界面如图 16-14 所示。当用户点击屏幕上的列表项时，会以对话框的形式弹出当前选中的联系人的信息，如图 16-15 和图 16-16 所示。



图 16-14　动态加载出的联系人列表



图 16-15　点击联系人名称时弹出的对话框



图 16-16　点击列表项右侧按钮时弹出的对话框

本范例虽然看起来简单没有涉及什么新的内容，但却非常难以掌握，尤其是在 jQuery Mobile 中动态地加载页面控件一直令许多新手感到困扰。在此将对范例中第 14~30 行的代码进行详细讲解。

首先是在 JavaScript 中为对象使用数组的方法，在前面已经提到过在 JavaScript 中实际是不存在类的，但是在 JavaScript 中却可以将任何一个元素都当作对象来使用。因此，在需要为对象定义数组时，只需要声明一个数组，然后像正常使用对象一样去使用其中的每一个元素就可以了，如范例第 19~21 行所示。但是要注意在第 18 行中还有一句代码：

```
people[i] = new Object();
```

如果没有它的存在，是不能将元素 people[i]当作对象来对每个属性进行赋值的。

生成了联系人信息之后，要做的就是如何将这些信息在屏幕上显示出来，找到范例 16-3 中的第 21~24 行代码如下所示：

```
<li>
<a>联系人</a>
<a></a>
</li>
```

它的作用是声明列表中的一个列表项，也是列表中实际被显示出来的部分，在该范例中要做的实际上也就是将这部分的内容使用脚本加载到页面当中去。除此之外，为了给用户的操作加入响应，比如当用户点击列表项时进行拨打电话的操作，还要对这段代码做一点小小的修改。修改后的内容如下：

```
<li>
<a onClick="call(1)">联系人</a>
<a onClick="message(1)"></a>
</li>
```

其中 call 和 message 分别是对用户操作进行响应的自定义函数，见范例中第 36~50 行所示。而为了让系统能够知道用户到底选择了哪个列表则需要为这两个函数加入一个参数，即上面代码中的 1，在实际使用时，该数字与相应联系人的 id 属性对应，由系统进行分配。

在范例的第 25~31 行，就是在 JavaScript 中生成这段代码时所使用的字符拼接方法，然后可

以使用 jQuery 提供的 append()方法将它们加入到列表中去（如范例第 31 行所示）。如果仅仅是这样，看到的结果就一定会是如图 16-17 所示的界面。

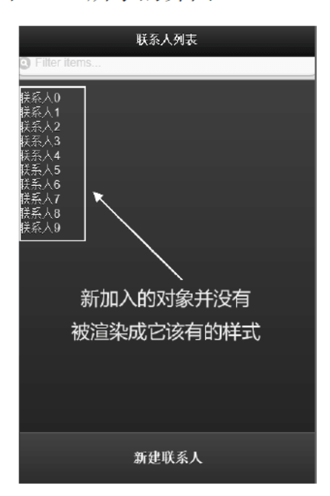


图 16-17　新加入的元素并没有被渲染成该有的样式

这是由于 jQuery Mobile 实际上就是一组 CSS 样式，但是在开发者使用它时不需要知道这些样式具体是哪一个，而是由 jQuery Mobile 的脚本在页面被加载时动态地进行分配的。但是当页面被加载完毕之后，如果页面上的内容发生了改变，jQuery Mobile 是不知道的，因此也不会对新加入的元素进行渲染。

那该怎么办呢？一个比较笨的办法是直接到 jQuery Mobile 的 CSS 文件中找到与列表项对应的样式，然后利用 JavaScript 对它进行人工加载。聪明的开发者应该会查阅文档或者是阅读本章的内容之后，使用本范例第 32 行所使用的方法对被改变的元素样式进行刷新。



由于目前使用 jQuery Mobile 开发的应用大多是基于 Web 的，可以利用后台的脚本来实现刷新，因此可能用不到这种对空间样式进行刷新的功能。但是当利用 jQuery Mobile 和 Cordova 进行本地应用的开发时，由于开发者可以依靠的只剩下 JavaScript 了，这种刷新的技能就变得非常重要。

16.5.2　页面的整合

以上已经实现了对联系人的动态加载，那么本小节要将范例进行一次整合，将它们合并到同一个页面中去。实现的方法如范例 16-7 所示。

【范例 16-7】最终实现的完整功能界面。

```
01  <!DOCTYPE html>
02  <html>
03  <head>
04  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

```
05  <title>使用 jQuery Mobile 实现联系人列表</title>
06  <meta name="viewport" content="width=device-width, initial-scale=0.5">
07  <!-- 引入 jQuery Mobile 所需的 CSS 样式文件 -->
08  <link rel="stylesheet"
href="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css" />
09  <!-- 引入 jQuery 脚本文件用来提高对 jQuery Mobile 的支持 -->
10  <script src="http://code.jquery.com/jquery-1.11.1.min.js"></script>
11  <!-- 引入 jQuery Mobile 所需的脚本文件 -->
12  <script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-
1.4.5.min.js"></script>
13  <script>
14  var people = new Array();                        // 新建一个数组用于存放联系人信息
15  var contact_Id = 0;
16  function get_contacts() {
17      for( i=0; i<10; i++) {
18      // 在实际开发中，将这部分换成利用 API 读取联系人信息
19          people[i] = new Object();                // 将数组元素类型转化为一个对象
20          people[i].id = i;
21          people[i].name = "联系人" + i;
22          people[i].number = "0411" + i + i;
23      }
24      show_List();
25  }
26  function show_List() {
27      for(i=0; i<people.length; i++) {
28          // 加入内容 <li><a onclick="call(i);">联系人</a><a
onclick="message(i);">
</a></li>
29          var str = ' <li>';
30          str = str + '<a onclick=\"call(' + i + ');\">';
31          str = str + people[i].name;
32          str = str + '</a>';
33          str = str + '<a onclick=\"message(' + i + ');\">';
34          str = str + '</a></li>';
35          $("#mylist").append(str);
36          $("#mylist").listview("refresh");
37      }
38  }
39  // 当程序完成时，在此处调用打电话的 API
```

```
40  function call(i) {
41      i++;
42      var str = "你想给第" + i + "个人打电话,";
43      str = str + "他叫: " + people[i-1].name + ",";
44      str = str + "他的电话号码是:" + people[i-1].number + "。";
45      alert(str);
46  }
47  // 通过此处切换到短信编辑界面
48  function message(i) {
49  //contact_Id = people[i].id;
50  //document.getElementById("sendTo").innerHTML = "发送给" + people[i].name;
51      window.plugins.CordovaPlugin.send(null,null,people[i-1].number,null)
52  }
53  // 当程序完成时，在此处调用发短信的 API
54  function send() {
55      var content = document.getElementById("message").value;
56      if(content != "") {
57          contact_Id++;
58          var str = "你想给第" + contact_Id + "个人发短信,";
59          str = str + "他叫: " + people[contact_Id-1].name + ",";
60          str = str + "他的电话号码是:" + people[contact_Id-1].number + "。";
61          alert(str);
62          alert(content);
63      }else {
64          alert("短信内容不能为空");
65      }
66  }
67  // 此处完成新建联系人的操作
68  function create() {
69      var name = document.getElementById("name").value;
70      var number = document.getElementById("number").value;
71      if( name != "" && number != "" ) {
72          var count = people.length;
73          people[i] = new Object();                // 将数组元素类型转化为一个对象
74          people[count].id = count;
75          people[count].number = number;
76          people[count].name = name;
77          var str = ' <li>';
78          str = str + '<a onclick=\"call(' + count + ');\">';
```

```
79          str = str + people[count].name;
80          str = str + '</a>';
81          str = str + '<a onclick=\"message(' + count + ');\">';
82          str = str + '</a></li>';
83          $("#mylist").append(str);
84          $("#mylist").listview("refresh");
85          location.href="#contact_list";
86      }
87  }
88  </script>
89  </head>
90  <body onLoad="get_contacts();">
91      <!-- 联系人列表界面 -->
92      <div data-role="page" data-theme="a" id="contact_list">
93          <div data-role="header" data-position="fixed">
94            <h1>联系人列表</h1>
95          </div>
96          <!-- 这里是联系人列表 -->
97          <ul id="mylist" data-role="listview" data-split-icon="gear" data-
filter
="true">
98              <!-- 新加入的内容将在这里被显示出来 -->
99          </ul>
100         <div data-role="footer" data-position="fixed">
101             <div  data-role="navbar" data-position="fixed">
102                 <ul>
103                     <li><a href="#contact_create"><h2>新建联系人</h2></a></li>
104                 </ul>
105             </div>
106         </div>
107     </div>
108     <!-- 新建联系人界面 -->
109     <div data-role="page" data-theme="a" id="contact_create">
110         <div data-role="header" data-position="fixed">
111             <a href="#contact_list" data-role="button">返回</a>
112               <h1>新建联系人</h1>
113          </div>
114         <div data-role="content">
115             <label for="basic">联系人姓名:</label>
```

```
116             <input type="text" id="name" placeholder="请输入联系人姓名" />
117             <label for="basic">联系人号码:</label>
118             <input type="tel" id="number" placeholder="请输入联系人号码"/>
119         </div>
120         <div data-role="footer" data-position="fixed">
121             <div  data-role="navbar" data-position="fixed">
122                 <ul>
123                     <li><a onClick="create();"><h2>创建联系人</h2></a></li>
124                 </ul>
125             </div>
125         </div>
127     </div>
128 </body>
129 </html>
```

从范例代码中不难看出，实际上本范例的主要部分就是将 3 个范例的主要 HTML 代码全部复制到了本范例的 body 标签中。其中第 92~107 行来自范例 16-8 的第 54~69 行，是联系人列表界面；第 51 行是我们直接调用我们刚才自定义插件的发送短信的接口；第 108~127 行来自于范例 16-6 的第 15~33 行，用于作为新建联系人界面来显示。但是在运行该页面后的界面与图 16-13 没有什么区别，只显示了联系人列表界面。那么页面中新加入的内容都到哪里去了呢？

这是由于 jQuery Mobile 在对页面进行渲染时，会默认只显示页面中 id 为 home 的那个页面，但是在本范例的第 92 和 109 行可以看到，在此次改动中为每一个 page 控件加入了一个 id，却并没有找到 id 值为 home 的页面。此时，jQuery Mobile 会默认只渲染第一个页面。

当需要时可以通过链接的方式在各个页面之间进行切换，如第 85 行和第 103 行所示。

除此之外，还可以看到在此次页面合并中，笔者新加入了一些页面的响应，如第 123 行的 onClick 属性等，用于为下一步加入 API 留下接口。除此之外，还在页面中对脚本进行了添加和修改。目前所实现的界面功能如下：

实现了联系人列表的动态加载，如图 16-14 所示。本范例将 get_contacts()方法进行了修改，仅保留了生成联系人数据的部分而将原本显示联系人的部分封装到了函数 show_List 中。

第 48~52 行处的 message 函数获取了被用户选中的联系人的 id，并将其保存到变量 contact_Id 中，然后修改短信编辑页面中顶部栏的内容，并切换至该界面。

当用户编辑完短信后，可以点击屏幕底部的发送按钮，此时会执行在第 54~66 行处的函数 send。通过阅读此处代码可以看出，该函数正是通过获取 contact_Id 的值来判断要将短信发送给哪个联系人的。

此外，还实现了新建联系人以及在新建联系人之后返回联系人列表，并将新建的联系人显示在屏幕上的功能，如图 16-18 和图 16-19 所示。而此时如果点击联系人列表项则能够获得该联系人的信息，如图 16-20 所示，该部分功能在范例中的第 68~87 行所示。

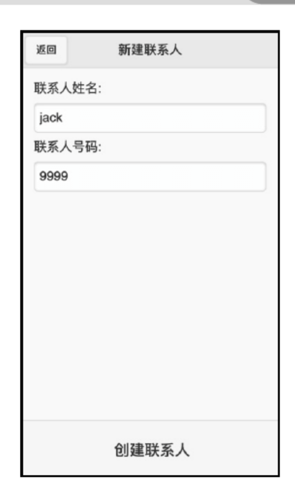


图 16-18　在新建联系人界面新建联系人





图 16-19　可以看到新创建的联系人被显示出来　　图 16-20　点击列表项则会显示新创建的联系人信息

16.6 最终功能的实现

通过以上的努力，可以说本项目已经实现得差不多了，现在我们要做的就是刚才代码中 45 行的：

```
45      alert(str);
```

替换成：

```
45  window.plugins.CordovaPlugin.send(null,null,people[i-1].number,null)
```

然后在手机机上测试一下看看有没有达到预想的效果。

16.7 小结

本章不但补齐了书中没有介绍的 Cordova 插件实现方法，还在项目中使用了 jQuery Mobile，因此本章对于初学者来说是很有学习价值的。除此之外，读者在学习完本章内容后还要深入思考一些更加复杂的问题，比如怎么解决 Cordova 获取联系人效率的问题，以及如何将本地存储应用到项目中。至此，本书的实战项目就全部结束了。